Comprando esta edição tenha acesso na WEB a:
+260 vídeo-aulas e +140 artigos

www.clubedelphi.net Feita para desenvolvedores

ClubeDelphi

Ano 6 · Edição 81 · R$8,90
DevMedia GROUP

ISSN 1517990-7

Especial

Integração via WebServices com o CEP dos Correios

Saiba como acessar o WS dos Correios e implemente na sua aplicação:
Preenchimento automático de endereço através do CEP
Busca de CEP através do endereço e vice-versa
Busca de endereço através de palavras-chave

POO

Uma abordagem prática no desenvolvimento de aplicações OO no Delphi

Na WEB AJAX com DELPHI

Confira no portal ClubeDelphi PLUS
4 vídeo aulas sobre AJAX com Delphi!
(tempo: **74minutos de vídeo**)

JVCL
Componentes free para validação, notificação popup e barra lateral estilo Outlook, dicas tipo balão e muito mais

Serviços do Windows
Crie serviços para o Windows e rode aplicações em background

Firebird
Novidades da versão 2.0

Stored Procedures
Crie funções para manipulação de strings

Nunit
Deixe sua aplicação .NET mais estável com teste unitários

Quantas cópias de seu software existem no mercado?

Esta é a chave mais segura do mundo contra pirataria de software.

Comprove você mesmo!

SafeNet®

Alameda Tocantins, 280 - Barueri-SP

Tel: +55 11 4208-7700

NÃO PERCA

PORTAL DO ASSINANTE

A ClubeDelphi tem uma novidade para você que comprou este exemplar na banca de jornal: você pode acessar GRATUITAMENTE, o Portal do Assinante ClubeDelphi!

Confira o que você encontra no Portal do Assinante:

- Mais de 220 Vídeo Aulas!
- Diversos Mini-Cursos Gratuitos!
- 1 Livro Eletrônico sobre ADO.NET e BDP!
- Mais de 140 Artigos Exclusivos!

Para Utilizar o Portal do Assinante, acesse www.devmedia.com.br/clubedelphi/portal.asp e utilize as informações abaixo:

Login: DVM.PL

Senha: B22FC

O acesso é válido por 30 dias a partida da data de lançamento da revista. Todos os meses a ClubeDelphi lhe dará uma senha válida para acessar o portal. Comprando a revista regularmente em bancas, você terá acesso ininterrupto a ele!

Corpo Editorial

Diretor Editorial e Administrativo
Gladstone Matos
gladstone@neoficio.com.br

Designer
Vinicius O. Andrade
viniciusoandrade@gmail.com

Capa
Antonio Xavier
jaxs_design@yahoo.com.br

Articulistas desta edição
Adriano Santos, Rodrigo Sendin, Michael Benford, Ivan Prenosil, Jorge Luis Bublitz, João Carlos da Silva, Marcelo S. Daibert e Marco Antônio P. Araújo

Editor Geral
Guinther Pauli
guinther@devmedia.com.br

Editor Técnico
Luciano Pimenta
lucianopimenta@clubedelphi.net

Revisão
Fabio Correa
fabiocorrea@devmedia.com.br

Distribuição
Fernando Chinaglia Dist. S/A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

Publicidade
Para informações sobre veiculação de anúncio na revista ou no site entre em contato com:
Kaline Dolabella
publicidade@devmedia.com.br

Atendimento ao Leitor

A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

Thiago Andrade – Atendimento ao Leitor
www.devmedia.com.br/central/default.asp
(21) 2220-5375

Kaline Dolabella – Gerente de Marketing e Atendimento
kalined@terra.com.br
(21) 2220-5375

Parcerias

Para fechar parcerias ou ações específicas de marketing com a DevMedia, entre em contato com:
Jeff Wendell
jeff@devmedia.com.br

ASSESPRO PARANÁ

Fale com o Editor!

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão! Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Guinther Pauli - Editor da Revista
guinther@devmedia.com.br

Luciano Pimenta - Editor do Site
lucianopimenta@clubedelphi.net

ClubeDelphi

Ano 6 - 81ª Edição - 2006 - ISSN 1517990-7 **Impresso no Br**

Editorial

A matéria de capa desta edição vai para um assunto que está em alta p os desenvolvedores Delphi: POO (na prática e em aplicações reais). M co Antônio, João Carlos e Marcelo Santos unem em uma só aplicação principais recursos da POO, incluindo herança, classes, encapsulamen mapeamento O/R (objeto relacional), persistência e muito mais.

O Adriano Santos mostra como criar serviços do Windows, ideal para cr suas aplicações Delphi que precisam rodar em background, processando segundo plano. Na VCL, fazemos isso facilmente através da classe TService Michael Benford selecionou 20 componentes para mostrar em seu artigo, fantástica biblioteca JVCL. Veja como é simples exibir notificações na barra tarefas (estilo Outlook), salvar dados no executável (em runtime), exibir hi na forma de balão, validar controles e muito mais!

O Ivan mostra como criar diversas Stored Procedures para manipu strings no Firebird. Marco Antônio e Marcelo Santos apresentam o NUn utitário que acompanha o Delphi e é destinado à criação de testes unitár em ambiente .NET (equivalente ao DUnit do mundo Win32). Curioso s bre as novidades da última versão do Firebird? O Jorge mostra os princip recursos do Firebird 2.0, e de praxe dá excelentes dicas de como você po migrar hoje seu banco de dados para a nova versão do banco open sour

Você certamente já deve ter, pelo menos uma vez na vida, preenchi um formulário de cadastro na Web. Em cadastros básicos, temos o cam CEP. Que tal criar um formulário Web que conecta diretamente no site c correios, obtém os dados para o CEP desejado e apresenta o resultado tempo real para o usuário? É o que mostra o Rodrigo Sendin.

Bom leitura e sucesso com o Delphi

Guinther Pauli – guinther@devmedia.com.br
Microsoft Certified: MCP, MCAD, MCSD.NET,
Borland Certified: Delphi 6, 7, 2005, 2006, Web, Kylix

Presente!

Como ter acesso a palestra online

Para comemorar a edição número 80 da revista ClubeDelphi, disponibilizamos para nossos leitores, assinantes e de bancas, uma palestra especial de Guinther Pauli sobre as novidades do Delphi 2006;

Acesse o link abaixo e faça o download da palestra:

www.devmedia.com.br/clubedelphi/ed80palestra

ÍNDICE

05 - Uma abordagem prática no desenvolvimento de aplicações OO no Delphi
João Carlos da Silva, Marcelo S. Daibert e Marco Antônio P. Araújo

12 - Criando serviços do Windows
Adriano Santos

17 - NUnit - um Framework de testes para .NET
Marcelo S. Daibert e Marco Antônio P. Araújo

22 - Integrando sua aplicação com a busca de CEPS dos Correios
Rodrigo Sendin e Luciano Pimenta

30 - Conheça na prática alguns componentes da fantástica JVCL
Michael Benford

43 - Criando Stored Procedures para manipular strings
Ivan Prenosil

46 - Novidades do Firebird 2.0
Jorge Luis Bublitz

Portal ClubeDelphi PLUS

+260 vídeo aulas

www.clubedelphi.net/portal

Caro Leitor,

O portal ClubeDelphi PLUS é a continuação, na Web, da revista ClubeDelphi. O portal recebe um conteúdo novo todo dia e hoje conta com: i) mais de 260 vídeo aulas; ii) diversos mini cursos gratuitos; iii) 1 livro eletrônico gratuito, de Guinther Pauli, sobre ADO.NET e BDP; iv) mais de 140 artigos exclusivos (que não foram publicados na revista)!;

Acesse o portal ClubeDelphi PLUS e receba muito mais conteúdo sobre Delphi! E o que é melhor: de graça! Todo leitor da revista ClubeDelphi, seja ele assinante ou comprador da revista em bancas, tem acesso ao portal (para quem compra em bancas, o acesso é válido por 30 dias).

Se você é assinante, utilize o seu login e senha pessoais para acessar o portal. Se você comprou em bancas, utilize o login e senha publicados na página do editorial desta edição.

Confira a seguir as últimas novidades do portal!

Boa leitura e sucesso!

Equipe DevMedia

Vídeo-Aulas

Performance em ASP.NET com ADO.NET

Guinther Pauli mostra nessa série de aulas, como obter o máximo de perfomance utilizando ADO.NET em aplicações ASP.NET.

Minicurso SysCash - Parte VII a IX

Veja nessa vídeo aula de Everson Volaco, mas um capítulo que mostra o sistema SysCash.

Componentes JEDI

Luciano Pimenta mostra várias aulas sobre essa maravilhosa suíte de componentes open source, o JEDI. Veja como trabalhar com componentes fáceis, controles de validação, semelhante aos que temos no ASP.NET.

Instalando o Firebird 2.0 junto com Firebird 1.5

Veja nessa vídeo aula de Paulo Quicoli, como instalar a nova versão do Firebird.

Não perca o mini curso de Ajax com Delphi 2006

Fabricio Desbessel iniciou um mini curso onde mostra como trabalhar com a tecnologia Ajax no Delphi 2006 com o MagicAjax.

Não perca as aulas sobre Fast Report

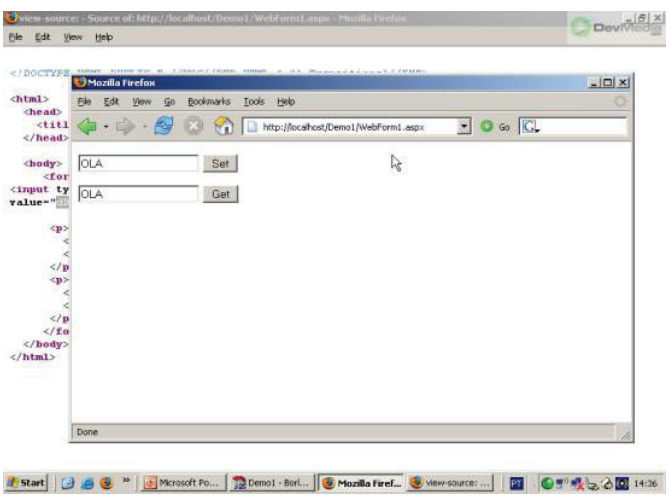
Paulo Quicoli mostra várias caracteristicas desse ótimo gerador de relatórios, chamado Fast Report. Veja como gerar etiquetas, mestre/detalhe, listagem e muito mais.

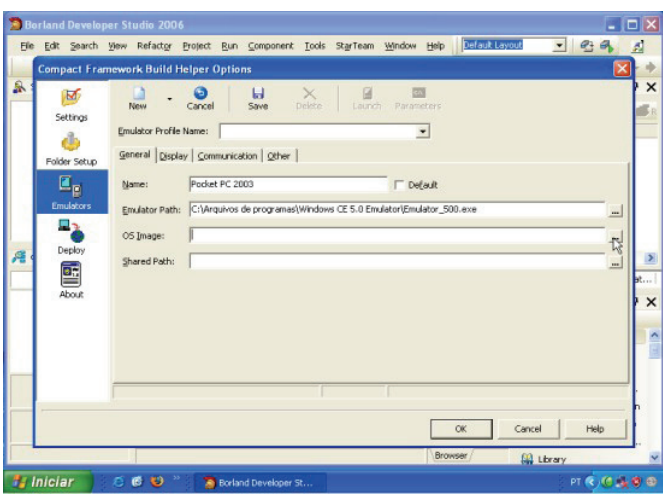
Compact Framework no Delphi - Parte II e III

Veja nessa vídeo aula de Jefferson Luis, como trabalhar com o Compact Framework no Delphi.

Word no Delphi

Veja nessa vídeo aula de Jefferson Junglaus, como criar documentos do Word, através de uma aplicação Delphi.





Desenvolvimento de aplicações OO no Delphi

Uma abordagem prática - Parte 1

JOÃO CARLOS DA SILVA
(jcsilva@pos.granbery.edu.br)
é Graduado em Sistemas de Informação pela Faculdade Metodista Granbery e Pós-Graduando em Ciência da Computação pela Universidade Federal de Viçosa.

MARCELO SANTOS DAIBERT
(mdaibert@npq.granbery.edu.br)
é Graduado em Sistemas de Informação pela Faculdade Metodista Granbery e Pós-Graduando em Ciência da Computação pela Universidade Federal de Viçosa.

MARCO ANTÔNIO P. ARAÚJO
(maraujo@granbery.edu.br)
é Professor do Curso de Bacharelado em Sistemas de Informação da Faculdade Metodista Granbery, Doutorando e Mestre em Engenharia de Sistemas e Computação pela COPPE/UFRJ, Especialista em Métodos Estatísticos Computacionais e Bacharel em Informática pela UFJF, Analista de Sistemas da Prefeitura de Juiz de Fora.

Com o passar dos anos, o paradigma Orientado a Objetos tem sido extensivamente utilizado na construção de sistemas de software e tornou-se um dos padrões no desenvolvimento de aplicações. São vários os benefícios que esse paradigma traz à equipe de desenvolvimento e talvez a isso se deva o seu sucesso.

No entanto, para realmente fazer uso dos benefícios da Orientação a Objetos, como reutilização de código, manutenibilidade do sistema, entre outros, é necessário que a equipe tenha domínio de assuntos co-relacionados como padrões de projeto, da linguagem de programação utilizada, persistência e principalmente entender os conceitos da Orientação a Objetos.

Atualmente, um dos obstáculos no desenvolvimento de aplicações OO é a persistência de objetos. Os bancos de dados puramente Orientados a Objetos são de fato os mais adequados para a persistência, mas a indisponibilidade atual desses, seja devido ao custo, diversidade e principalmente amadurecimento, faz com que seja necessária uma busca por alternativas para a realização dessa tarefa.

Uma das soluções encontradas para o problema é a utilização de camadas de persistência para manipulação dos objetos utilizando bancos de dados relacionais. Essa abordagem vem sendo amplamente utilizada no mercado, principalmente para o desenvolvimento de sistemas de médio a grande porte.

Basicamente une a rapidez e o amadurecimento das bases de dados relacionais com os benefícios da Programação Orientada a Objetos (POO). A função principal de uma camada de persistência é portar transparentemente os objetos de uma aplicação para uma base de dados relacional de forma genérica.

Para isso, são utilizadas principalmente técnicas de mapeamento objeto-relacional com o intuito de mapear as classes e associações para tabelas e relacionamentos. A utilização de uma camada de persistência, embora pareça trivial inicialmente, se mostra complexa na prática, uma vez que são várias as configurações que

devem ser feitas para que a camada funcione corretamente, além de outros fatores que dificultam sua utilização.

Utilizando esse mesmo raciocínio, uma outra solução a abordada neste artigo, é a própria aplicação acessar o banco de dados relacional e realizar o mapeamento objeto-relacional, persistindo seus objetos. Com isso, a aplicação é capaz de persistir e manipular os objetos em bancos de dados relacionais de forma rápida e transparente, não necessitando de uma camada de persistência desenvolvida por terceiros.

Buscando entender essa abordagem, o objetivo deste artigo é exemplificar o uso prático do paradigma Orientado a Objetos no desenvolvimento de uma aplicação no ambiente Delphi 7, utilizando persistência em um banco de dados relacional.

Para isso, são apresentados alguns padrões de desenvolvimento que buscam maximizar os benefícios da utilização do paradigma e facilitam a tarefa de persistência de objetos.

Este artigo está dividido em duas partes. Na primeira é apresentado um estudo de caso utilizado para exemplificar o desenvolvimento de software orientado a objetos utilizando os padrões de desenvolvimento, além da definição das classes, métodos e atributos do modelo. É apresentado ainda o conceito de mapeamento objeto-relacional e como utilizar esta abordagem no escopo do estudo de caso apresentado. Por fim, ainda na primeira parte é definida a estratégia de conexão com o banco de dados e das classes responsáveis pela persistência a manipulação dos objetos.

Na segunda parte será implementada a estratégia de persistência de objetos utilizada, além de exemplos e codificação de persistência e manipulação dos objetos.

Estudo de Caso

No escopo deste artigo é apresentado um estudo de caso para exemplificar o desenvolvimento de um software Orientado a Objetos utilizando padrões de desenvolvimento no ambiente Delphi 7. Com ele é possível verificar e exemplificar a utilização de padrões de projeto que auxiliam o desenvolvimento da aplicação, de técnicas de Programação Orientadas a Objetos e de persistência de objetos.

O estudo de caso, apresenta um fragmento de um sistema maior, representado aqui pela relação entre funcionários e departamentos de uma empresa, através das classes *Departamento*, *Funcionario*, *FuncionarioMensalista* e *FuncionarioDiarista*. A classe *Funcionario* é abstrata, servindo para a definição dos elementos comuns às suas subclasses através do mecanismo de herança.

Todos os funcionários devem estar lotados em um *Departamento*, que pode ter vários funcionários. A **Figura 1** representa o diagrama de classes do estudo de caso.

Através desse estudo de caso, são abordados temas como classes, objetos, métodos, atributos, persistência de objetos, herança, polimorfismo, associações e padrões de projeto, além de temas pertinentes ao ambiente de desenvolvimento, como utilização de componentes, classes e controles disponibilizados pelo Delphi.

Para a implementação do estudo de caso será utilizado o Delphi 7 como ferramenta de desenvolvimento e, para a persistência de objetos será utilizado o banco de dados relacional

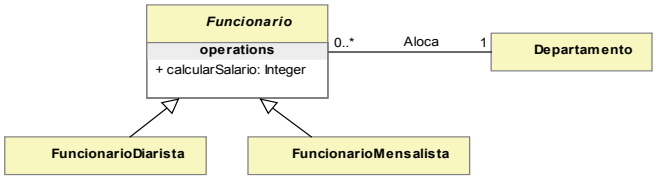


Figura 1. Diagrama de Classes do estudo de caso

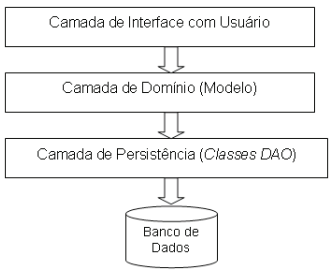


Figura 2. Disposição das camadas do estudo de caso

Firebird. É utilizada ainda a abordagem de programação em camadas, que busca, entre outras coisas, o desacoplamento de funções da aplicação, facilitando assim o desenvolvimento e garantindo maior manutenibilidade.

No escopo deste artigo é utilizada uma camada para apresentação, representada pela interface do sistema, uma camada de domínio, representada pelas classes de negócio da aplicação, uma camada de persistência, representada pelas classes DAO (*Data Access Object*) que são responsáveis pela persistência e manipulação dos objetos no banco de dados.

E, por fim, uma camada de banco de dados, representado pelo SGBD (Sistema Gerenciador de Banco de Dados) Firebird. A **Figura 2** exibe a disposição das camadas utilizadas no estudo de caso.

Classes, Atributos e Métodos

Com um projeto criado no Delphi 7, o primeiro passo é implementar a definição das classes do modelo apresentado. Para isso, é criada uma unit chamada *untDominio.pas*. Neste artigo, todas as classes do modelo são definidas e implementadas nessa unit.

A **Listagem 1** apresenta a definição da classe *Departamento* e sua implementação. Já na **Listagem 2** é apresentada a classe abstrata *Funcionario*. Por fim, a **Listagem 3**, exibe a implementação da classe *FuncionarioDiarista*. A definição e implementação da classe *FuncionarioMensalista* não é apresentada pois é muito similar à *FuncionarioDiarista*.

Na **Listagem 1**, nas linhas 4 e 5 são definidos os atributos privados da classe *Departamento*. O primeiro é o atributo OID (*Object Identificator*), que representa o identificador do objeto no modelo. No modelo relacional, esse atributo é considerado o campo chave-primária. O segundo atributo é a descrição do departamento.

Entre as linhas 7 e 14 são apresentados os métodos da classe. Os primeiros são os métodos *get* e *set*, responsáveis por manipular os valores dos atributos, mantendo assim o encapsulamento da classe.

Nota: Encapsulamento é a prática de limitar o acesso às informações de um objeto, fazendo com que so-

mente os seus métodos tenham acesso aos dados e à manipulação dos atributos. O encapsulamento disponibiliza o objeto com toda sua funcionalidade sem a necessidade de saber internamente o seu funcionamento, aumentando assim a manutenibilidade e reutilização do sistema.

Os demais métodos definidos são os responsáveis pela manipulação dos objetos da classe no banco de dados. Eles basicamente fazem acesso às classes de persistência, que são definidas posteriormente neste artigo, e manipulam os objetos no banco de dados.

Para isso são definidos os seguintes métodos: *Persistir*, salva e atualiza os objetos, *Obter*, recupera um objeto específico na base de dados, *ObterTodos*, recupera todos os objetos da classe e, por fim, o *Destruir*, responsável pela exclusão de um determinado objeto.

Listagem 1. Definição da classe Departamento

```
1.type
2.  Departamento = class
3.private
4.  OID: string;
5.  fDescricao: string;
6.public
7.  function getOID(): string;
8.  procedure setOID(pOID: string);
9.  function getDescricao: string;
10.  procedure setDescricao(pDescricao: string);
11.  function Persistir: Boolean;
12.  class function Obter(pOID: string): Departamento;
13.  class function ObterTodos: TList;
14.  function Destruir: Boolean;
15.end;
16.
17. implementation
18.
19.function Departamento.getOID: string;
20.begin
21.  result := OID;
22.end;
23.procedure Departamento.setOID(pOID: string);
24.begin
25.  OID := pOID;
26.end;
27.function Departamento.getDescricao: string;
28.begin
29.  result := fDescricao;
30.end;
31.procedure Departamento.setDescricao(pDescricao: string);
32.begin
33.  fDescricao := pDescricao;
34.end;
35.function Departamento.Persistir: Boolean;
36.var
37.  objPersistente: DepartamentoDAO;
38.begin
39.  objPersistente:= DepartamentoDAO.Create;
40.  result:=objPersistente.Persistir(self);
41.  objPersistente.Free;
42.end;
43.class function Departamento.Obter(pOID: string): Departamento;
44.var
45.  objPersistente: DepartamentoDAO;
46.begin
47.  objPersistente := DepartamentoDAO.Create;
48.  result := objPersistente.Obter(pOID);
49.  objPersistente.Free;
50.end;
51.class function Departamento.ObterTodos: TList;
52.var
53.  objPersistente: DepartamentoDAO;
54.begin
55.  objPersistente := DepartamentoDAO.Create;
56.  result := objPersistente.ObterTodos;
57.  objPersistente.Free;
58.end;
59.function Departamento.Destruir: Boolean;
60.var
61.  objPersistente: DepartamentoDAO;
62.begin
63.  objPersistente := DepartamentoDAO.Create;
64.  result:= objPersistente.Destruir(self.getOID);
65.  objPersistente.Free;
66.end;
```

Entre as linhas 35 e 42, é apresentado o *Persistir*. Ele inicialmente define o uso de uma variável do tipo *DepartamentoDAO*, que é a implementação responsável pela persistência e comunicação da aplicação com a base de dados, para a classe *Departamento*.

Após instanciado, na linha 39, a variável *objPersistente* invoca o seu *Persistir*, passando como parâmetro o *Departamento* em memória (*self*). Caso seja obtido sucesso na tarefa, a função então retorna *true*, caso contrário, *false*. Ao final, na linha 41, o objeto é destruído.

A linha 43 apresenta a assinatura do método de classe *Obter*, que recebe como parâmetro o OID do objeto a ser recuperado. Esse método retorna o objeto recuperado da base de dados. Nas linhas 51 a 58, é apresentado o método de classe *ObterTodos*.

Esse retorna um *TList* dos objetos *Departamento* da base de dados. Já entre as linhas 59 a 66, é definido o *Excluir*, que é responsável pela exclusão do objeto da base de dados, retornando *true* caso obtenha êxito, ou *false* em caso de falha.

Nota: Método de Classe é um método que não há a necessidade de se instanciar um objeto da classe para poder acessá-lo. Diferentemente do método de instância, que como o próprio nome diz, é necessário a instanciação do objeto para sua invocação.

Na classe *Departamento*, o *Obter* e *ObterTodos* são métodos de classe, já que para realizar essas tarefas não é necessário o objeto instanciado na memória. Para definir um método de classe no Delphi, basta utilizar o modificador *class* na definição da assinatura do método. Caso não seja utilizado esse modificador, o compilador trata-o como um método de instância.

Já na **Listagem 2**, entre as linhas 2 e 21 é definida a classe *Funcionario* com seus atributos e métodos. Entre as linhas 24 e 82 são implementados os métodos da classe. Nas linhas 4 a 6 são definidos os atributos privados da classe, sendo acessados pelos métodos *get* e *set* apresentados entre as linhas 34 e 57.

A classe possui os atributos *OID*, *fNome* e *fDepartamento*. Esse último é a implementação da associação apresentada no modelo, no qual um funcionário é lotado em um *Departamento*. Na classe ainda são definidos os métodos *Persistir*, *ObterTodos*, *ObterFuncionariosPorOIDDepartamento*, *Destruir* e *calcularSalario*.

O *Persistir* é virtual e abstrato, o que significa que ele não possui implementação na classe, devendo ser codificado nas classes filhas. Da mesma forma, o *calcularSalario* também é virtual e abstrato já que o mesmo é redefinido nas subclasses, de acordo com as necessidades específicas de implementação de cada especialização.

Isso é uma das características de um conceito chamado polimorfismo, que é a capacidade de métodos de mesma assinatura comportar-se de maneiras diferentes, em uma mesma hierarquia. Nas linhas 25 a 28 é implementado o método construtor da classe *Funcionario* e das linhas 29 à 33 o destrutor.

O objetivo é permitir a instanciação do *Departamento* no momento da instanciação de um objeto *Funcionario*, como apresentado na linha 27. Para o destrutor o objetivo é destruir o objeto *Departamento* associado juntamente com o objeto *Funcionario*. Os demais métodos definidos entre as linhas 58 a 82 são os de manipulação e persistência da classe.

Esses, assim como os definidos na classe *Departamento* apresentada, fazem chamada à classe *FuncionarioDAO* de acesso aos dados do funcionário.

Por fim, na **Listagem 3**, a classe *FuncionarioDiarista* é apresentada. Logo na linha 2, a classe é definida especificando a sua superclasse *Funcionario*. Além de herdar todos os atributos da superclasse, o funcionário diarista apresenta o atributo *fNumDias*, representando o número de dias trabalhados em um mês e o atributo *fValorDia*, representando o valor recebido por dia trabalhado.

Listagem 2. Definição da Classe Funcionario

```
1.type
2.  Funcionario = class
3.private
4.  OID: string;
5.  fNome: string;
6.  fDepartamento: Departamento;
7.public
8.  constructor Create;
9.  destructor Destroy;
10.  function getOID: string;
11.  procedure setOID(pOID: string);
12.  function getNome: string;
13.  procedure setNome(pNome: string);
14.  function getDepartamento: Departamento;
15.  procedure setDepartamento(pDepartamento: Departamento);
16.  function Persistir: Boolean;virtual;abstract;
17.  class function ObterTodos: TList;virtual;
18.  class function ObterFuncionariosPorOIDDepartamento(
       pOIDDepartamento: string): TList;
19.class function Destruir(pOID: string): Boolean;virtual;
20.function calcularSalario: Real;virtual;abstract;
21.end;
22.
23.implementation
24.
25.constructor Funcionario.Create;
26.begin
27.  fDepartamento := Departamento.Create;
28.end;
29.destructor Funcionario.Destroy;
30.begin
31.  fDepartamento.Free;
32.  inherited;
33.end;
34.function Funcionario.getOID: string;
35.begin
36.  result := OID;
37.end;
38.procedure Departamento.setOID(pOID: string);
39.begin
40.  OID := pOID;
41.end;
42.function Funcionario.getNome: string;
43.begin
44.  result := fNome;
45.end;
46.procedure Funcionario.setNome(pNome: string);
47.begin
48.  fNome := pNome;
49.end;
50.function Funcionario.getDepartamento: Departamento;
51.begin
52.  result := fDepartamento;
53.end;
54.procedure Funcionario.setDepartamento(pDepartamento: Departamento);
55.begin
56.  fDepartamento := pDepartamento;
57.end;
58.class function Funcionario.ObterTodos: TList;
59.var
60.  objPersistente: FuncionarioDAO;
61.begin
62.  objPersistente := FuncionarioDAO.Create;
63.  result := objPersistente.ObterTodos;
64.  objPersistente.Free;
65.end;
66.class function Funcionario.ObterFuncionariosPorOIDDepartamento(
67.pOIDDepartamento: string): TList;
68.var
69.  objPersistente: FuncionarioDAO;
70.begin
71.  objPersistente := FuncionarioDAO.Create;
72.  result := objPersistente.ObterFuncionariosPorOIDDepto(pOIDDepartamento);
73.  objPersistente.Free;
74.end;
75.class function Funcionario.Destruir(pOID: string): Boolean;
76.var
77.  objPersistente: FuncionarioDAO;
78.begin
79.  objPersistente := FuncionarioDAO.Create;
80.  result:= objPersistente.Destruir(pOID);
81.  objPersistente.Free;
82.end;
```

Listagem 3. Definição da Classe FuncionarioDiarista

```
1.type
2.  FuncionarioDiarista = class (Funcionario)
3.private
4.  fNumDias: Integer;
5.  fValorDia: Currency;
6.public
7.function getNumDias: Integer;
8.procedure setNumDias(pNumDias: Integer);
9.function getValorDia: Currency;
10.procedure setValorDia(pValorDia: Currency);
11.function Persistir: Boolean; override;
12.class function Obter(pOID: string): FuncionarioDiarista;
13.class function ObterTodos: TList; override;
14.function Destruir: Boolean;reintroduce; overload;
15.function calcularSalario: Real; override;
16.end;
17.
18.implementation
19.
20.function FuncionarioDiarista.getNumDias: Integer;
21.begin
22.  result := fNumDias;
23.end;
24.procedure FuncionarioDiarista.setNumDias(pNumDias: Integer);
25.begin
26.  fNumDias := pNumDias;
27.end;
28.function FuncionarioDiarista.getValorDia: Currency;
29.begin
30.  result := fValorDia;
31.end;
32.procedure FuncionarioDiarista.setValorDia(pValorDia: Currency);
33.begin
34.  fValorDia := pValorDia;
35.end;
36.function FuncionarioDiarista.Persistir: Boolean;
37.var
38.  objPersistente: FuncionarioDiaristaDAO;
39.begin
40.  objPersistente:= FuncionarioDiaristaDAO.Create;
41.  result:= objPersistente.Persistir(self);
42.  objPersistente.Free;
43.end;
44.class function FuncionarioDiarista.Obter(
     pOID: string): FuncionarioDiarista;
45.var
46.  objPersistente: FuncionarioDiaristaDAO;
47.begin
48.  objPersistente := FuncionarioDiaristaDAO.Create;
49.  result := objPersistente.Obter(pOID);
50.  objPersistente.Free;
51.end;
52.class function FuncionarioDiarista.ObterTodos: TList;
53.var
54.  objPersistente: FuncionarioDiaristaDAO;
55.begin
56.  objPersistente := FuncionarioDiaristaDAO.Create;
57.  result := objPersistente.ObterTodos;
58.  objPersistente.Free;
59.end;
60.function FuncionarioDiarista.Destruir: Boolean;
61.var
62.  objPersistente: FuncionarioDiaristaDAO;
63.begin
64.  objPersistente := FuncionarioDiaristaDAO.Create;
65.  result := objPersistente.Destruir(self.getOID);
66.  objPersistente.Free;
67.end;
68.function FuncionarioDiarista.calcularSalario: Real;
69.begin
70.  result := fNumDias * fValorDia;
71.end;
```

A classe apresenta a definição e implementação do *Persisitir* (linhas 36 a 43), abstrato na superclasse, além dos métodos *Obter*, *ObterTodos*, *Destruir* e *calcularSalario*. O *Persistir* é responsável por persistir e/ou atualizar os objetos na base de dados. O *Obter* tem a funcionalidade de recuperar um objeto na base de dados.

O *ObterTodos* tem a função de recuperar todos os objetos do tipo *FuncionarioDiarista* e o *Destruir* de excluir o objeto referenciado da base de dados. Por fim, o *calcularSalario*, implementa a operação de cálculo de salário para um funcionário diarista, como apresentado a partir da linha 68.

Mapeamento Objeto-Relacional e banco de dados

O mapeamento objeto-relacional é uma abordagem que permite a construção de sistemas utilizando o paradigma Orientado a Objetos com a persistência desses objetos em bancos de dados relacionais. Utilizando-se de técnicas e estratégias específicas, é possível mapear classes com seus atributos e associações para o modelo relacional.

Utiliza-se uma abstração bastante intuitiva no sentido de que uma classe pode ser mapeada para uma tabela no banco de dados relacional e atributos da classe para campos da tabela. Porém, algumas diferenças entre os dois modelos, como o OID, tipos de dados, herança e associações, demandam um estudo mais detalhado das estratégias de mapeamento.

Para o estudo de caso, a construção da base de dados seguiu esse raciocínio. A classe *Departamento* foi mapeada para a tabela *departamento*, com seus atributos sendo os campos da tabela. A hierarquia *Funcionario* foi mapeada para uma única tabela *funcionario*, com todos os atributos (como campos da tabela), acrescido de um campo chamado *tipo*, que representa a classe que instanciou cada objeto.

Existem outras duas possíveis alternativas: criar uma nova tabela para cada classe da hierarquia e relacioná-las utilizando o conceito de chave estrangeira ou criar uma nova tabela para cada classe concreta, contendo os seus atributos, mais os atributos da classe abstrata. Neste artigo isso não foi feito em função da simplicidade do modelo.

Assim, a **Listagem 4** apresenta o *script* para criação do banco, resultante do mapeamento objeto-relacional realizado com as classes do estudo de caso.

Listagem 4. Script para a criação das tabelas no banco de dados

```
CREATE TABLE DEPARTAMENTO (
  OID        CHAR(38) NOT NULL,
  DESCRICAO  VARCHAR(50));

CREATE TABLE FUNCIONARIO (
  OID           CHAR(38) NOT NULL,
  NOME          VARCHAR(50),
  DEPARTAMENTO  CHAR(38),
  TIPO          CHAR(1) NOT NULL,
  VALORMES      DECIMAL(13,2),
  VALORDIA      DECIMAL(13,2),
  NUMDIAS       INTEGER);

ALTER TABLE DEPARTAMENTO ADD CONSTRAINT
  PK_DEPARTAMENTO PRIMARY KEY (OID);
ALTER TABLE FUNCIONARIO ADD CONSTRAINT
  PK_FUNCIONARIO PRIMARY KEY (OID);
ALTER TABLE FUNCIONARIO ADD CONSTRAINT
  FK_FUNCIONARIO FOREIGN KEY (DEPARTAMENTO)
  REFERENCES DEPARTAMENTO (OID);
```

Com a base de dados criada no Firebird, é possível configurar a forma de acesso da aplicação ao banco. O acesso aos dados é realizado por meio de um Data Module. Nesse estão presentes os componentes que permitem a realização da conexão com banco de dados, recuperação e atualização de dados por meio de expressões em SQL.

A **Figura 3** mostra os componentes que fazem parte do Data Module denominado *dtmdlDados*.

O *sqlcnctnDados* é do tipo *TSQLConnection*, que permite através de drivers específicos estabelecer conexão com bancos de dados. Neste exemplo é utilizado o driver do InterBase, que também é compatível com a versão do Firebird que foi utilizada nessa aplicação.

Os parâmetros de conexão utilizados na aplicação são apresentados na **Figura 4.**

Os demais componentes: *sqldtstPrincipal*, *dtstprvdrPrincipal* e *clntdtstPrincipal* são do tipo *TSQLDataset*, *TDataSetProvider* e *TClientDataSet*. São ligados ao *sqlcnctnDados* bem como interligados entre si por meio das propriedades *SQLConnection*, *DataSet* e *ProviderName*, respectivamente.

Tais componentes são responsáveis por manipular conjuntos de registros recuperados e destinados ao banco de dados.

Classes de Persistência

As classes de persistência e manipulação de objetos são tratadas em uma unit separada, no exemplo chamada de *untPersistencia*. Dessa forma, isolam-se em uma camada todas as questões relativas à persistência e, por conseqüência, questões relativas ao acesso a banco de dados. Isso facilita manutenções futuras na aplicação, além de obter outras vantagens do desenvolvimento em camadas.

Essas classes representam um padrão de projeto chamado *Data Access Object,* que, basicamente, define o encapsulamento do acesso aos dados por meio de classes especificamente construídas para essa tarefa. Assim, para cada classe persistente do modelo existe uma classe DAO correspondente.

Sendo assim, as classes DAO são responsáveis por utilizar o Data Module definido anteriormente, para realizar a manipulação dos dados. O Data Module não é referenciado diretamente, mas sim instanciados objetos a partir desse.

A **Figura 5** exibe a estrutura de classes da *untPersistente*, onde são implementadas as classes *TPersistente* e as classes DAO correspondentes às classes de domínio.

A hierarquia de classes apresentada na **Figura 5** demonstra uma forma possível de implementar classes DAO que tratam da persistência de objetos. Uma outra forma possível, não tratada neste artigo, seria o uso de interface no lugar da classe *TPersistente*, o que evitaria a construção hierarquizada das classes DAO.

De uma forma ou de outra, as classes DAO implementam somente métodos responsáveis pela recuperação e manutenção dos dados armazenados em banco de dados, de suas correspondentes classes de domínio. A classe *TPersistente*, que é ancestral das demais classes, possui implementação em apenas um método (*GerarOID*).

Os demais são abstratos, caracterizando essa classe como uma

classe abstrata, exigindo a implementação em suas classes descendentes. Dessa forma garante-se que as classes DAO possuam implementações básicas de persistência e manipulação de objetos, através dos métodos *Persistir*, *Obter*, *ObterTodos* e *Destruir*.

A **Listagem 5** apresenta a definição da classe *TPersistente*.

A classe apresenta ainda os métodos *Persistir*, *Obter*, *ObterTodos* e *Destruir*, que além de serem abstratos, sofrem sobrecarga nas classes descendentes. Já o *GerarOID*, possui implementação, visto que o OID (identificador do objeto) possui uma regra única para sua formação.

É utilizado o algoritmo GUID, disponível no Delphi para formação do OID. Isso garante a unicidade do OID gerado. Outros algoritmos podem ser utilizados aqui caso seja necessário, bem como algoritmos diferentes podem ser implementados para cada classe DAO. O GUID foi escolhido por ser de fácil implementação.

O identificador gerado, embora seja um conjunto de caracteres do sistema hexadecimal como, por exemplo *{81B35AC9-F6C5-411E-9176-3D0A4D2C8E04}*, não será problema visto que o usuário não necessita ter contato direto com ele. O OID é utilizado para identificar um objeto bem como atuar como chave primária em tabelas do banco de dados relacional.

A **Listagem 6** apresenta o código correspondente à implementação do *GerarOID*.

A **Listagem 7** mostra a definição das demais classes DAO.

Na definição das classes DAO, nota-se principalmente a presença dos métodos *Persistir*, *Obter*, *ObterTodos* e *Destruir*. Tais métodos redefinem a implementação dos métodos abstratos da classe *TPersistente* e são os responsáveis pela manipulação dos objetos na base de dados relacional. Percebe-se que *Persistir* e *Obter* possuem assinaturas diferentes da classe ancestral, por isso utilizam a diretiva *overload*.

A classe *FuncionarioDAO* é ancestral das classes *FuncionarioDiaristaDAO*, espelhando o domínio. Para essa hierarquia de classes de funcionário foi definida uma única tabela no banco de dados para armazenamento das informações, tanto de funcionários diaristas quanto funcionários mensalistas. Outras formas também seriam possíveis como uma tabela para cada classe da hierarquia ou então uma tabela para cada classe concreta.

Da forma que estão definidas as classes da hierarquia de *FuncionarioDAO*, nem todos métodos são implementados em todas as classes. Por exemplo, a classe *FuncionarioDAO* não possui implementação de *Persistir* e *Obter*, visto que cada caso (mensalista e diarista) são tratados separadamente.

Já o *Destruir* possui implementação na classe *FuncionarioDAO* e não possui nas classes descendentes, pois a exclusão de um funcionário independe do seu tipo. Ainda, o *ObterTodos* está implementado em todas as classes da hierarquia, tendo semântica diferente. Na classe *FuncionarioDAO* o método recupera todos os objetos de funcionários independente do tipo, e nas classes descendentes recupera todos os objetos do respectivo tipo.

Além dos métodos padrões definidos na classe *TPersistente*, a classe *FuncionarioDAO* implementa o *ObterFuncionariosPorOIDDepto*, que permite a recuperação de objetos de funcionários a partir do identificador de um departamento, que está associado.

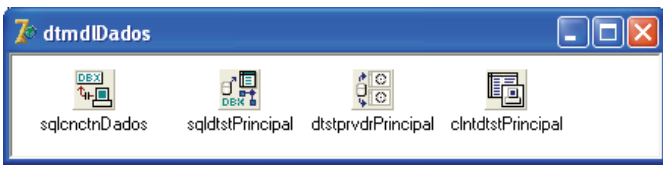


Figura 3. Data Module de conexão ao banco de dados

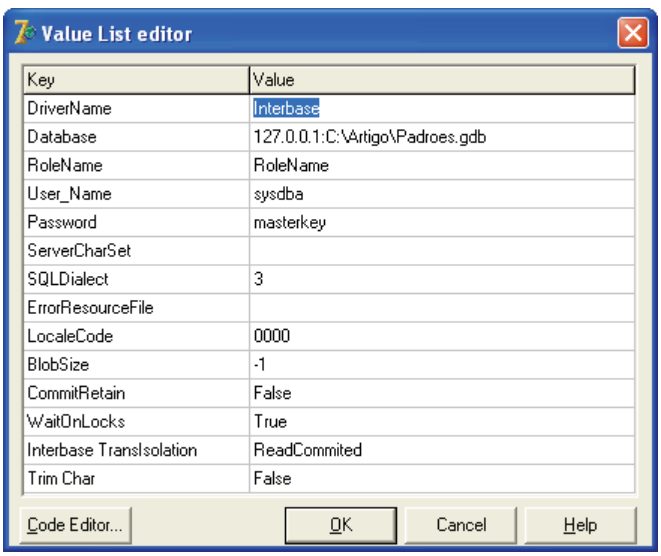


Figura 4. Configuração do sqlcnctnDados

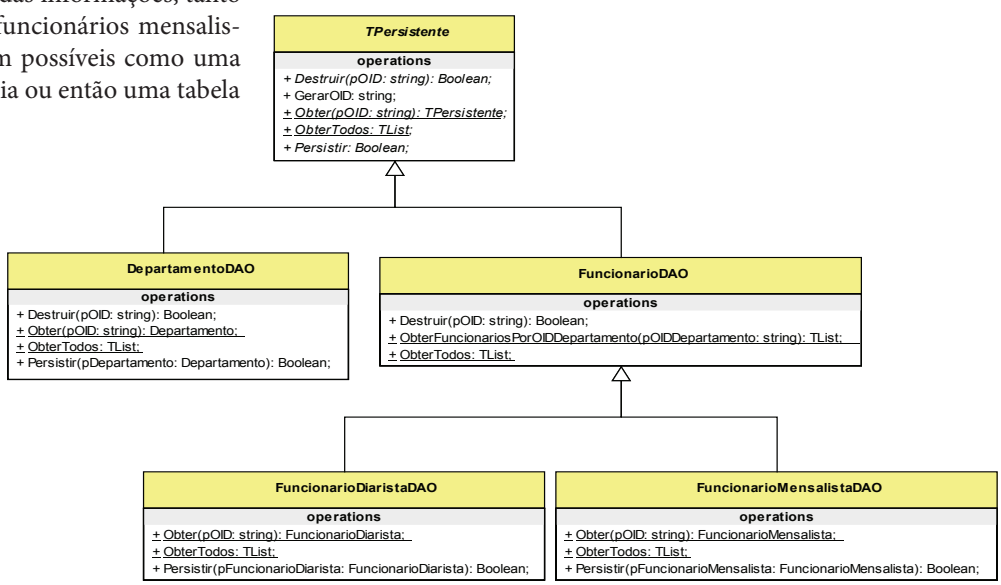


Figura 5. Estrutura de classes responsáveis pela persistência dos objetos

Listagem 5. Definição da Classe TPersistente

```
TPersistente = class (TObject)
public
  function GerarOID: string;
  function Persistir: Boolean; virtual; abstract;
```

Listagem 6. Código-fonte do Método GerarOID

```
function TPersistente.GerarOID: string;
var
  vGUID: TGUID;
begin
  CreateGUID(vGUID);
  result := GUIDToString(vGUID);
end;
  class function Obter(pOID: string): TPersistente;
virtual; abstract;
  class function ObterTodos: TList; virtual;
abstract;
  function Destruir(pOID: string): Boolean; virtual;
abstract;
end;
```

Listagem 7. Definição das classes DAO

```
DepartamentoDAO = class (TPersistente)
public
  function Persistir(pDepto: Departamento): Boolean;
    reintroduce;overload;
  class function Obter(pOID: string): Departamento;
    reintroduce;overload;
  class function ObterTodos: TList;override;
  function Destruir(pOID: string): Boolean; override;
  end;
FuncionarioDAO = class (TPersistente)
public
  class function ObterTodos: TList;override;
  class function ObterFuncionariosPorOIDDepto(
    pOIDDepartamento: string): TList;
  function Destruir(pOID: string): Boolean; override;
  end;
FuncionarioDiaristaDAO = class (FuncionarioDAO)
public
  function Persistir(
    pFuncDia: FuncionarioDiarista): Boolean;
    reintroduce;overload;
  class function Obter(
    pOID: string): FuncionarioDiarista;
    reintroduce;overload;
  class function ObterTodos: TList;override;
end;
```

Conclusão

Esta primeira parte do artigo teve como objetivo abordar de forma prática a utilização do paradigma Orientado a Objetos no contexto do desenvolvimento de software no ambiente Delphi e de técnicas aliadas, como mapeamento objeto relacional e utilização de padrões de desenvolvimento.

Foi possível observar na prática a criação de classes e a definição de atributos e métodos. Foi abordado também o tema de mapeamento objeto-relacional, que é de suma importância quando da persistência de objetos utilizando um banco de dados relacional. Ao final, foi definida a estratégia utilizada para conexão com o banco de dados e foi apresentada a definição das classes responsáveis pela manipulação e persistência dos objetos (Classes DAO). Foi possível também definir o padrão de projeto DAO (*Data Access Object*), que busca estreitar os benefícios da orientação a objetos com a maturidade do banco de dados relacional.

Na próxima edição, será apresentada a implementação das classes de persistência com o objetivo de persistir e manipular os objetos no banco de dados relacional, além de encapsular toda a comunicação com o banco de dados. Serão apresentados alguns exemplos práticos no contexto do estudo de caso aqui apresentado e definido. ■

ClubeDelphi PLUS

Acesse agora mesmo o Portal do Assinante ClubeDelphi e assista a uma vídeo-aula de Luciano Pimenta que mostra como usar um driver free e específico do Firebird para a tecnologia dbExpress.

www.devmedia.com.br/articles/viewcomp.asp?comp=1941

ClubeDelphi PLUS

Acesse agora mesmo o portal do assinante ClubeDelphi e assista a uma vídeo-aula de Guinther Pauli que mostra como utilizar GUID para gerar chaves primárias para o banco de dados através do Delphi.

www.devmedia.com.br/articles/viewcomp.asp?comp=3321

PENSE...

QUANTO TEMPO
VOCÊ GASTARIA
PARA DESENVOLVER
COBRANÇA COM BOLETOS
BANCÁRIOS PARA
APENAS UM BANCO
NO SEU SOFTWARE

COBREBEMX

- 56 BANCOS E MAIS DE 430 CARTEIRAS DE COBRANÇA PARA IMPRESSÃO E/OU ENVIO DE BOLETO BANCÁRIO POR EMAIL;
- GERAÇÃO DE BOLETOS ON LINE;
- GERAÇÃO E LEITURA DE ARQUIVOS (REMESSA/RETORNO) NOS PADRÕES FEBRABAN E CNAB;
- MAIS DE 40 EXEMPLOS EM DIVERSAS LINGUAGENS DE PROGRAMAÇÃO

Cobre Bem Tecnologia

DOWNLOADS E INFORMAÇÕES EM WWW.COBREBEM.COM

Criando serviços do Windows

ADRIANO SANTOS
(artes@doiscliques.com)
é desenvolvedor Delphi desde 1998. Possui sólidos conhecimentos em linguagem SQL e nos bancos de dados MySQL, MSSQL e FireBird. Professor de informática e programador PHP. É colunista nos portais SQL Magazine, ClubeDelphi e membro da Comissão Editorial da revista ClubeDelphi.

A entrada maciça dos sistemas operacionais da Microsoft baseados em tecnologia NT (*New Technology*), como por exemplo: Windows XP, 2000 e 2003, trouxe para o desenvolvedor Win32 uma vantagem sem igual em relação aos antigos SO's: os Windows Services ou Serviços do Windows. Com eles é possível desenvolver sistemas automatizados que realizam determinadas tarefas continuamente ou em momentos pré-agendados, com/sem a interação do usuário.

Um exemplo disso é o próprio Windows, que aproveita a sua própria tecnologia para efetuar diversas tarefas, tais como: Instalação de Programas, Horário do Windows, Atualizações Automáticas, entre outras. Todas elas encontradas em *Painel de Controle>Ferramentas de Sistema>Serviços*.

Neste artigo vamos elaborar um pequeno sistema, capaz de efetuar automaticamente a cópia de um banco de dados IB/FB todos os dias, em um horário pré-agendado.

Porque usar Serviços?

Se olharmos um pouco para o passado, mais especificamente para os antigos sistemas operacionais, veremos que era bastante complicado desenvolver tarefas automatizadas que rodassem em tempo real e que funcionassem mesmo que o usuário não *logasse* na rede. Essa, certamente, é uma das principais vantagens dos serviços.

Podemos determinar que o servidor faça uma checagem de rotina no banco de dados a cada *n* horas sem intervenção do usuário. Se dispuséssemos apenas de Windows 95 ou 98, teríamos que desenvolver um aplicativo que executado continuamente, faria nossa checagem, mas para isso o usuário deveria logar na rede para que o software fosse disparado e aberto, talvez com uma entrada no registro do Windows ou uma ação de arquivo de lote (BAT). Isso, com certeza, tornaria o aplicativo dependente do usuário.

Porém, se desenvolvermos um *Service*, o mesmo é adicio-

nado na lista de tarefas do Windows e executado assim que a máquina é ligada, e mesmo que o servidor esteja em modo de *standby*, o serviço continua funcionando.

Entendendo o projeto

Para entendermos melhor, vamos desenvolver uma aplicação básica que fará uma cópia simples do banco para uma nova pasta. O exemplo terá um arquivo de inicialização (INI) com a Hora, Origem e Destino da cópia (você pode adaptar para verificar juntamente os minutos). Uma *thread* interna "rodará" continuamente, interceptará o horário e efetuará a cópia, do arquivo GDB.

Existem *n* aplicabilidades para os serviços, mas para ser mais objetivo o exemplo cabe bem. Então, mão na massa.

Iniciando o projeto

Para iniciar uma nova aplicação, clique em *File>New>Other* e selecione a opção *Service Application* (**Figura 1**).

Note que o Delphi cria o escopo do novo projeto (arquivo DPR) como mostrado na **Listagem 1.**

Listagem 1. Novo projeto de serviço

```
program Project1;

uses
  SvcMgr,
  Unit1 in 'Unit1.pas' {Service1: TService};

{$R *.RES}

begin
  Application.Initialize;
  Application.CreateForm(TService1, Service1);
  Application.Run;
end.
```

Nesse momento podemos começar a desenvolver as funções e ações, mas antes vamos entender como funciona toda essa tecnologia. O serviço, em termos visuais, pode ser comparado a um "Data Module avançado".

Repare que o Delphi criou a *unit1* e nela temos apenas uma tela branca, lembrando bastante um Data Module. Nele é possível adicionar componentes não-visuais e codificá-los normalmente.

Um item interessante no "Data Module", são os eventos bem particulares, como por exemplo:

- *AfterInstall*, *AfterUnistall*, *BeforeInstall* e *BeforeUnistall*: interagem com a instalação e desinstalação do serviço;
- *OnContinue*: quando o serviço está pausado e volta a ser ativado;
- *OnStart*: quando o serviço é iniciado;
- *OnShutDown*: acontece quando o serviço é finalizado.

E outros: *OnPause*, *OnExecute*, *OnCreate*, *OnDestroy* e *OnStop*. Há ainda particularidades em suas propriedades que são:

- *DisplayName*: nome que aparecerá na lista de tarefas do Windows (para o nosso exemplo, digite "Automatizador de Backups");
- *Interactive*: se interage com o usuário;
- *AllowPause* e *AllowStop*: permite ou não que o serviço possa ser pausado ou parado;
- *StartType*: indica qual o procedimento de *start* do serviço.

Entendidas as principais características, salve seu projeto com o nome "aBackup.dpr". A *unit1* salve com o nome de "uServicoMain.pas" e nomeie o Data Module como "srvServico".

Funções e procedimentos

Vamos começar criando a *thread* que fará o trabalho de cópia do banco de dados. Clique em *File>New>Other* e escolha a opção *Thread Object* (**Figura 2**).

Em *Class Name* coloque o nome da nova classe como "ServicoThread" e clique em OK. Em seguida salve a *unit* como "uServicoThread.pas". O único procedimento que vamos codificar é o *Execute* da *Thread* (que já tem seu cabeçalho e declaração criado).

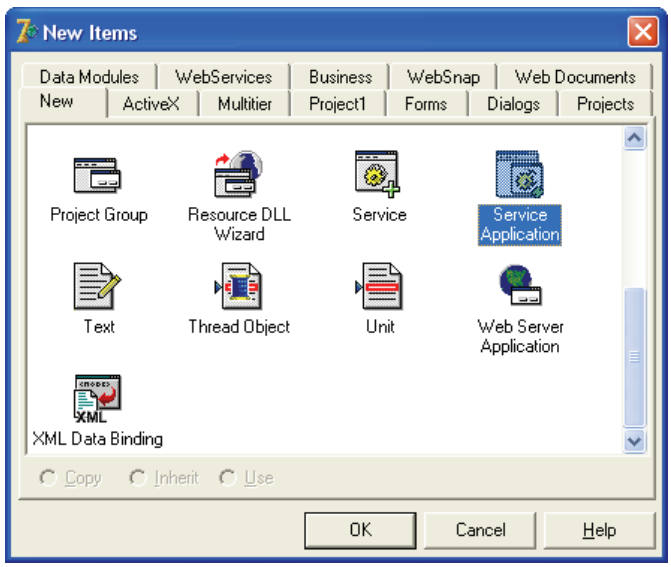


Figura 1. Criando uma aplicação Service Application

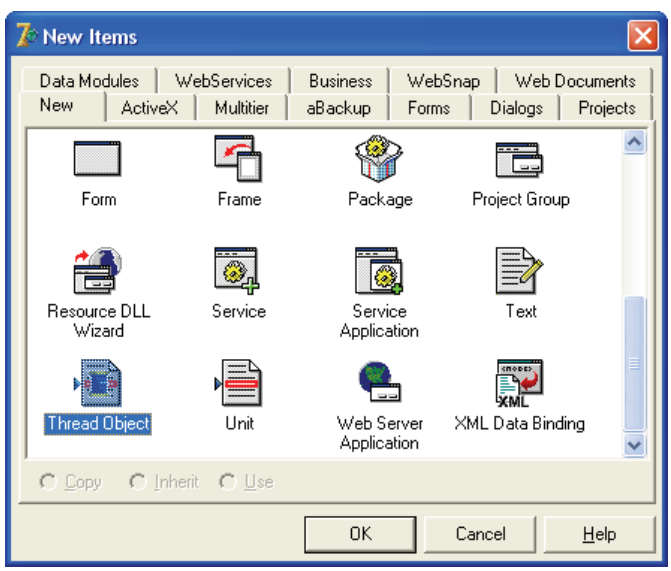


Figura 2. Criando um Thread Object

Sua função é trabalhar em tempo real infinitamente, até que uma ação seja requisitada, em nosso caso: *a cópia*. O código encontra-se na **Listagem 2**.

A primeira seqüência do código faz o carregamento das informações de Hora, Origem e Destino. Observe a **Listagem 3**.

Essas informações estão contidas no arquivo *config.ini* que tem o seguinte layout (crie o arquivo com esse nome no diretório da aplicação e adicione os valores a seguir):

```
[CONFIG]
Hora=01:00:00
Origem=C:\ClubeDelphi\ClubeDelphi.gdb
Destino=G:\Backups\
```

Logo em seguida, fazemos um *while* de acordo com a propriedade *Terminated*, que verifica se a hora atual é igual à hora da cópia, presente no arquivo INI. (**Listagem 4**)

Note que o *while* está dentro de um bloco *try/except*, porém não há ações para o *except*. Nesse ponto recomendo a criação de um *log* pra registrar quaisquer erros ocorridos durante o processo.

Retornando à unit principal

A *uServicoMain* fará o trabalho de criar e ativar a *thread*, fazendo com que o serviço aguarde a ação da cópia. Para isso altere as seções *public* e *private* como na **Listagem 5**.

O mais importante aqui é a criação das variáveis *fServicePri* e *fThreadPri*, e também as funções *ParaThread* e *IniciaThread*. Como o próprio nome sugere, as duas terão o objetivo de parar e iniciar a *thread* respectivamente.

Criamos também uma variável do tipo *TServicoThread*, ou seja, do mesmo tipo da classe *thread* (adicione no uses *uServicoThread*). Pressione CTRL+SHIFT+C para criar o cabeçalho dos métodos. As principais características da função *IniciaThread* são:

• *ServicoThread.Resume*: Inicia a *thread* fazendo com que o *Execute* seja disparado;

• *ServicoThread.Priority*: Determina a prioridade que a *thread* terá no sistema operacional diante das demais tarefas.

O código das *threads* pode ser visto na **Listagem 6**.

> **Nota:** Tome cuidado ao determinar serviços com alta prioridade, pois outras tarefas podem estar sendo deixadas de lado pelo Sistema Operacional.

ParaThread tem a finalidade de parar o processamento da *thread* e eliminar sua instância. Ela é acionada pelo *OnStop* do serviço. Primeiro verificamos se há uma instância rodando e em caso positivo a terminamos, fazendo uma chamada ao *Terminate*.

O *WaitFor* aguarda enquanto o sistema operacional pára o processo. Após isso basta acionarmos o *Free* para que seja destruída de memória nossa *thread*.

Listagem 2. Algorítimo de execução da Thread

```
interface

uses Forms, Windows, Messages, SysUtils, IniFiles,
  ExtCtrls, Dialogs, Classes;
...
procedure TServicoThread.Execute;
var
  Config: TIniFile;
  HoraBackup: TDateTime;
  Origem, Destino, nmBackup: string;
begin
  Config := TIniFile.Create(ExtractFilePath(
    Application.ExeName) + 'config.ini');
  with Config do
  begin
    HoraBackup := ReadTime('CONFIG', 'HORA',
      HoraBackup);
    Origem := ReadString('CONFIG', 'ORIGEM', Origem);
    Destino := ReadString('CONFIG', 'DESTINO',
      Destino);
    Config.Free;
  end;
  FreeOnTerminate := False;
  { Loop principal }
  try
    while not Terminated do
    begin
      if Time = HoraBackup then
      try
        nmBackup := FormatDateTime('YYYYMMDD', Now) +
          '.gdb';
        CopyFile(PChar(Origem), PChar(ExtractFilePath(
          Destino) + nmBackup), False);
      except
      end;
    end;
  except
    on E: Exception do ;
  end;
  Terminate;
end;
```

Listagem 3. Trecho de código que carrega as configurações do sistema de backup

```
Config := TIniFile.Create(ExtractFilePath(
  Application.ExeName) + 'config.ini');
with Config do
begin
  HoraBackup := ReadTime('CONFIG', 'HORA',
    HoraBackup);
  Origem := ReadString('CONFIG', 'ORIGEM', Origem);
  Destino := ReadString('CONFIG', 'DESTINO', Destino);
  Config.Free;
end;
```

Listagem 4. Trecho de código que efetua a cópia da base

```
while not Terminated do
begin
  if Time = HoraBackup then
  try
    nmBackup := FormatDateTime('YYYYMMDD', Now) +
      '.gdb';
    CopyFile(PChar(Origem), PChar(ExtractFilePath(
      Destino) + nmBackup), False);
  except
  end;
end;
```

Listagem 5. Seções private e public da unit principal

```
uses uServicoThread;
...
type
  TsrvBackup = class(TService)
  private
    fServicePri: Integer;
    fThreadPri: Integer;
    function ParaThread: Boolean;
    function IniciaThread(ThreadPri: Integer): Boolean;
  public
    ServicoThread: TServicoThread;
    ...
  end;
```

Listagem 6. Algorítimos da Thread

```
function TsrvBackup.ParaThread: Boolean;
begin
  Result := False;
  { Parando a thread }
  if Assigned(ServicoThread) then
  try
    ServicoThread.Terminate;
    { Se estiver suspensa, re-start com Resume }
    if ServicoThread.Suspended then
      ServicoThread.Resume;
    { Aguardando sua finalização }
    ServicoThread.WaitFor;
    ServicoThread.Free;
    ServicoThread := nil;
    Result := True;
  except
    on E: Exception do ;
  end else
  begin
    Result := True;
  end;
end;
function TsrvBackup.IniciaThread(
  ThreadPri: Integer): Boolean;
begin
  Result := False;
  { Criamos a thread e configuramos seus valores default }
  if not Assigned(ServicoThread) then
  try
    { Instancia a thread e determnina sua prioridade }
    ServicoThread := TServicoThread.Create(True);
    case ThreadPri of
      0: ServicoThread.Priority := tpIdle;
      1: ServicoThread.Priority := tpLowest;
      2: ServicoThread.Priority := tpLower;
      3: ServicoThread.Priority := tpNormal;
      4: ServicoThread.Priority := tpHigher;
      5: ServicoThread.Priority := tpHighest;
    end;
    ServicoThread.Resume;
    if not ServicoThread.Suspended then
      Result := True;
  except
    on E: Exception do ;
  end;
end;
```

No *IniciaThread*, primeiro verificamos se não há nenhuma instância da mesma criada usando o *Assigned*. Em caso positivo chamamos o *Create* e então determinamos a *Priority* (prioridade). Logo em seguida basta chamar *Resume* pra iniciar.

Note que novamente temos um bloco try/except sem nenhum tratamento de erro. Neste ponto recomendo a criação de um log registrando o erro caso ocorra. Na **Listagem 7** você encontra o código completo para todos os eventos do serviço.

O *ServiceController* e *GetServiceController* (implementados pelo Delphi) são usados pelo serviço para gerar um *handler*. Isso é necessário para dar o *start* no serviço. Sem a chamada ao *Controller* o mesmo não pode ser iniciado.

Temos também no código da listagem anterior, os eventos *OnStop* e *OnExecute*, que respectivamente chama *ParaThread*, que finaliza o processo, e associa o *handler* do serviço à *thread* de execução enquanto o serviço estiver ativo. Já o evento *OnStart* é responsável por iniciar a *thread*. Nele determinamos os valores *default* do serviço e tentamos iniciar a *thread* chamando *IniciaThread*. Se falhar, retornará um erro e então paramos a *thread* pra evitar maiores problemas.

Depois de pronto basta compilar o projeto e instalar. Para instalar/desinstalar basta usar as seguintes linhas de comando no *prompt* do DOS:

```
Instalação
C:\>aBackup.exe /install
Desinstalação
C:\>aBackup.exe /uninstall
```

Listagem 7. Código para Iniciar, Pausar e Parar a Thread

```
procedure ServiceController(CtrlCode: DWord); stdcall;
begin
  srvServico.Controller(CtrlCode);
end;

function TsrvServico.GetServiceController:
  TServiceController;
begin
  Result := ServiceController;
end;

procedure TsrvServico.ServiceExecute(Sender: TService);
begin
  { Evento OnExecute }
  { Loop enquanto o estiver ativo }
  while not Terminated do
    ServiceThread.ProcessRequests(False);
end;

procedure TsrvServico.ServiceStop(Sender: TService;
  var Stopped: Boolean);
begin
  { Evento OnStop }
  { Marca a thread como parada }
  Stopped := ParaThread;
end;

procedure TsrvServico.ServiceStart(
  Sender: TService; var Started: Boolean);
begin
  { Evento OnStart }
  { Aqui determinamos os valores valores padrão
    quando iniciado }
  Started := False;
  { Prioridade do Serviço }
  fServicePri := NORMAL_PRIORITY_CLASS;
  { Prioridade da Thread }
  fThreadPri := Integer(tpLower);
  { Aqui determinamos a prioridade do serviço }
  case fServicePri of
    0: SetPriorityClass(GetCurrentProcess,
         IDLE_PRIORITY_CLASS);
    1: SetPriorityClass(GetCurrentProcess,
         NORMAL_PRIORITY_CLASS);
    2: SetPriorityClass(GetCurrentProcess,
         HIGH_PRIORITY_CLASS);
    3: SetPriorityClass(GetCurrentProcess,
         REALTIME_PRIORITY_CLASS);
  end;
  { Tentamos iniciar a thread, se ter algum problema
    então paramos tudo }
  if IniciaThread(fThreadPri) then
    { Flag que marca o "sucesso" no start da thread }
    Started := True
  else
  begin
    { Marca a flag como false (erro) e manda
      parar a thread }
    Started := False;
    ParaThread;
  end;
end;
```

Na **Figura 3** você pode ver o serviço sendo instalado e iniciado.

Na **Figura 4** temos o serviço instalado no painel de Serviços do Windows.

Conclusão

O desenvolvimento de serviços torna o processo de automatização do sistema mais robusto e, ao mesmo tempo, simples. Com os serviços é possível inovar criando diversas ferramentas que complementam ainda mais sua aplicação, agregando valor e tecnologia. ■

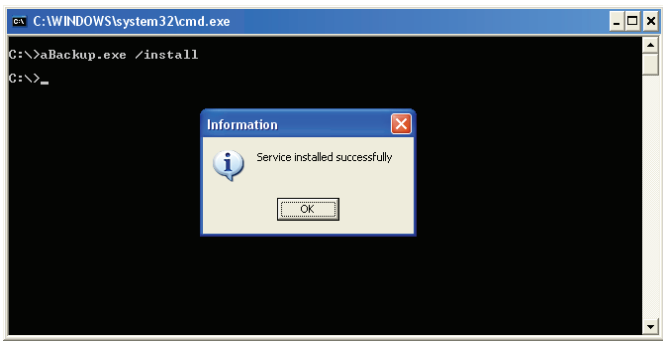


Figura 3. Instalando o serviço

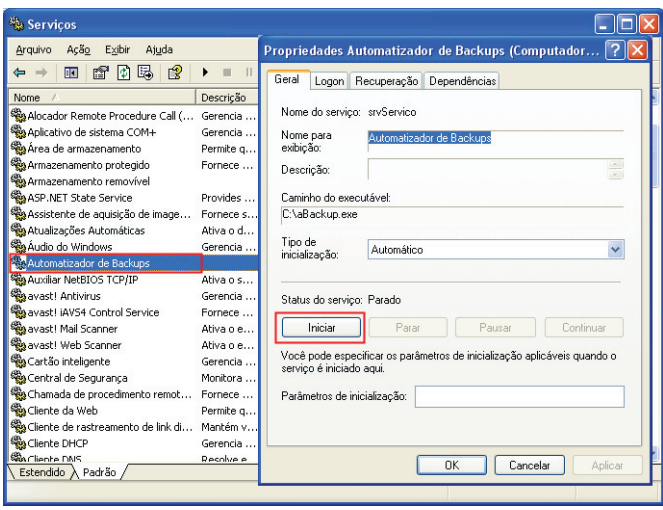


Figura 4. Serviço já instalado no Windows e pronto para iniciar

NUnit - um Framework de testes para .NET

MARCELO SANTOS DAIBERT
(mdaibert@npq.granbery.edu.br)
é Graduado em Sistemas de Informação pela Faculdade Metodista Granbery e Pós-Graduando em Ciência da Computação pela Universidade Federal de Viçosa, Desenvolvedor de Sistemas há 5 anos.

MARCO ANTÔNIO P. ARAÚJO
(maraujo@granbery.edu.br)
é Professor do Curso de Bacharelado em Sistemas de Informação da Faculdade Metodista Granbery, Doutorando e Mestre em Engenharia de Sistemas e Computação pela COPPE/UFRJ e Analista de Sistemas da Prefeitura de Juiz de Fora.

Uma das maiores preocupações atualmente é com relação à qualidade dos sistemas desenvolvidos e, consequentemente, à sua estabilidade no ambiente de produção. Falhas podem ocasionar problemas tanto para o desenvolvedor quanto para o cliente, muitas vezes gerando prejuízos financeiros e comprometendo os cronogramas firmados.

Assim, o processo de desenvolvimento deve considerar etapas de verificação da qualidade dos produtos de software, antecipando-se às falhas de forma a minimizar o número de defeitos nos sistemas. Sabe-se que é impossível garantir que todos os defeitos de um sistema foram eliminados, uma vez que o número de testes necessários para essa avaliação é normalmente em quantidade extremamente elevada, inviabilizando sua aplicação de forma manual.

Dessa forma, o teste automatizado torna-se imprescindível para viabilizar uma política de desenvolvimento baseado em testes. Para isso, existem ferramentas capazes de planejar e executar testes de forma automatizada. Nesse contexto, o *NUnit* é uma ferramenta de testes unitários para o ambiente de desenvolvimento .NET que basicamente carrega uma biblioteca (uma DLL compilada no .NET) com os testes planejados e os executam, apresentando os resultados para o desenvolvedor.

Para a criação e desenvolvimento do estudo de caso proposto neste artigo, é utilizado o Delphi 2006. No entanto, o *NUnit* é compatível com qualquer ambiente de desenvolvimento que gere binários .NET.

Configurando o ambiente de desenvolvimento

Inicialmente, é necessário configurar o ambiente para a geração da biblioteca DLL com os testes que são carregados pelo *NUnit*. Para isso, deve-se criar um projeto no Delphi 2006, através do menu *File>New>Other*.

A **Figura 1** apresenta a interface exibida com opções de elementos para serem adicionados ao projeto. Deve-se escolher a opção *Library* da categoria *Delphi for .NET Projects*.

A partir desse ponto, ao compilar o projeto, é gerada uma DLL e esse arquivo é o carregado pelo *NUnit* para a execução dos testes unitários. Salve o projeto com o nome de "nUnitLibrary". No projeto, deve-se adicionar uma referência ao *NUnit*. Com isso, a biblioteca torna-se capaz de acessar as funcionalidades disponibilizadas pelo *NUnit* para a criação de testes unitários.

No *Project Manager* clique com o botão direito no item *References* e escolha a opção *Add Reference*. Ao invocar essa opção é apresentada uma janela para escolha de referências a serem adicionadas no projeto.

A referência ao *NUnit* deve ser encontrada na aba *.NET Assemblies* com o nome *nunit.framework*, como exemplificado na **Figura 2**. Caso a mesma não se encontre disponível, é necessário verificar se o *NUnit* está realmente instalado ou adicionar a referência de forma manual.

> **Nota:** Baixe a última versão do *NUnit* no endereço: *www.nunit.org/index.php?p=download*. Sua instalação é simples e rápida. Lembre-se apenas de usar a versão para o .NET Framework 1.1.

Para fazer a referência manual, clique em *Browse* e localize o arquivo no disco. O nome do arquivo é *nunit.framework.dll* e está disponível dentro do diretório *bin* da pasta de instalação do *NUnit*.

Para finalizar, clique em *Add Reference* e OK. Depois de registrada a referência ao *NUnit*, a mesma é exibida na seção *References* no *Project Manager*, como exemplificado pela **Figura 3**. Com isso, o ambiente está configurado para a programação dos testes.

Criando os Testes

O estudo de caso proposto neste artigo apresenta uma classe chamada *TAluno*, com os atributos *frequencia*, *nota1*, *nota2* e *provafinal*. Essa classe possui dois métodos: o *setDados*, responsável por atribuir valores aos atributos e o *CalcularAprovacao*, responsável por verificar se o aluno foi aprovado ou não, de acordo com sua freqüência e suas notas.

A **Listagem 1** apresenta o código-fonte da classe, exibindo sua definição e implementação dos métodos. Crie um unit, dando o nome de "untClasses" e adicione o código da listagem.

O *CalcularAprovacao* verifica inicialmente se a freqüência do aluno é menor que 75%. Caso seja, o aluno é reprovado de imediato, fazendo com que a função retorne falso. Caso contrário é verificada a média da *nota1* com a *nota2*. Caso essa média for menor que 3, esse aluno também é reprovado. Caso contrário, se a média for maior que 7, o aluno é aprovado.

Se a média for maior que 3 e menor que 7, o aluno estará em prova final e será aprovado caso o valor da média de sua nota de prova final com a média anterior for superior a 5. Caso contrário será reprovado.

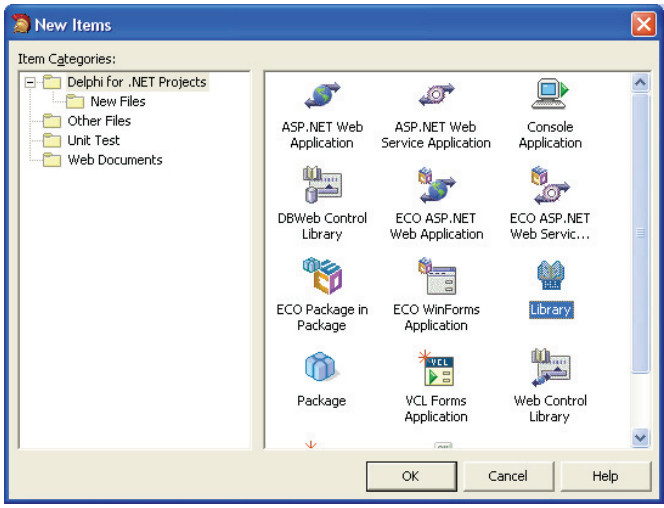


Figura 1. Adicionando uma Library ao projeto

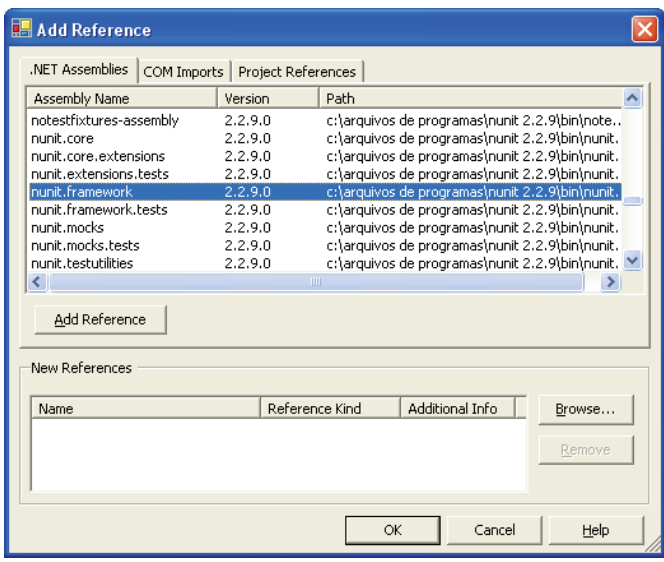


Figura 2. Adicionando uma referência ao NUnit

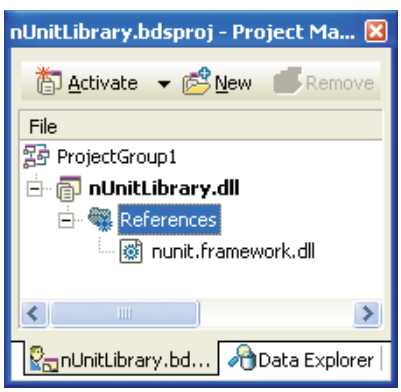


Figura 3. Referência ao NUnit adicionada

Listagem 1. Classe TAluno

```
unit Unit1;

interface

type
  TAluno = class
    private
      frequencia, nota1, nota2, provafinal: Real;
    public
      procedure setDados (pFrequencia, pNota1, pNota2,
        pProvaFinal: real);
      function CalcularAprovacao: boolean;
end;

implementation

function TAluno.CalcularAprovacao: boolean;
var
  vMedia: real;
begin
  if self.frequencia < 75 then
  begin
    Result := False;
  end
  else
  begin
    vMedia := (self.nota1 + self.nota2) / 2;
    if vMedia < 3 then
    begin
      Result := False;
    end
    else
    begin
      if vMedia > 7 then
      begin
        Result := True;
      end
      else
      begin
        if (vMedia + self.provafinal) / 2 < 5 then
        begin
          Result := False;
        end
        else
        begin
          Result := True;
        end;
      end;
    end;
  end;
end;

procedure TAluno.setDados(pFrequencia, pNota1,
  pNota2, pProvaFinal: real);
begin
  self.frequencia := pFrequencia;
  self.nota1 := pNota1;
  self.nota2 := pNota2;
  self.provafinal := pProvaFinal;
end;
```

O objetivo então é criar casos de teste para o *CalcularAprovacao* verificar essas regras. Para isso é necessário adicionar uma nova unit ao projeto, onde será construída uma classe de teste, com os casos de teste descritos como métodos dessa classe.

É importante verificar que a classe com os casos de teste faz referência direta à *TAluno*, sendo necessário então adicionar na seção *uses* uma referência à unit onde está definida a classe *TAluno* (*untClasses*). Outra classe importante a ser adicionada na seção *uses* é a *NUnit.Framework*, que contém os métodos e atributos disponibilizados pelo *NUnit* para a criação dos testes.

Crie uma nova unit dando o nome de "untTeste". A **Listagem 2** apresenta a codificação da classe de testes, contendo sua definição e implementação dos métodos *TesteAprovacao1* até *TesteAprovacao5*, que são os casos de teste.

Deve-se observar o uso do atributo *[TestFixture]* antes da definição da classe *TTeste* na linha 6. Esse atributo deve ser utilizado antes das classes definidas como classes de testes, para que o *NUnit* possa reconhecê-las dessa forma.

O *SetUp* é responsável pela inicialização do teste, como a instanciação do objeto, por exemplo. É importante destacar o uso do atributo *[SETUP]* antes da definição do método na linha 20. Esse atributo indica ao *NUnit* que o método definido a seguir deve ser executado antes de qualquer caso de teste.

Já o *TearDown* é responsável por finalizar o teste, como liberar a memória ocupada pelo objeto instanciado, por exemplo. Nesse é usado o atributo *[TEARDOWN]* como é exemplificado na linha 26. Esse atributo indica ao *NUnit* que o método definido a seguir deve ser executado ao final de todos os casos de teste.

Os demais métodos listados são os casos de teste. Para esses, é usado o atributo *[TEST]* antes da sua definição, indicando ao *NUnit* que o método a seguir é um caso de teste. No primeiro caso de teste é atribuído ao objeto do tipo *TAluno* a freqüência de 60% e o valor zero para os demais atributos. Como o aluno é reprovado com essa freqüência, não importam os valores das notas.

Essa situação pode ser observada na linha 35. Já na linha 36, é invocado o *IsFalse* da classe *Assert*. Essa classe é responsável por definir o valor esperado do teste para comparação com o valor obtido na sua execução. Nesse caso, o valor esperado para retorno do *CalcularAprovacao* é falso (*IsFalse*), uma vez que um aluno com 60% de freqüência é reprovado de imediato, testando assim a primeira condição da função, onde verifica se a freqüência é menor que 75%.

Na linha 42 é apresentada a definição do segundo caso de teste com um aluno de 90% de freqüência, primeira nota com valor 2, segunda nota com 1 e prova final com 0, pois não importa seu valor nessa situação. Para esse caso é esperado um retorno falso da função (linha 43).

Já na linha 49 é listado o terceiro caso de teste onde é configurado o objeto com 90% de freqüência, valores 9, 5 e 0 como primeira nota, segunda nota e prova final, respectivamente. O valor esperado de retorno para esse caso de teste é verdadeiro (*IsTrue*). Observe nesse caso, que a média é 7.

O quarto caso de teste é iniciado com 90% de freqüência, primeira e segunda notas com valor 4 e 5 e prova final com valor 4, como definido na linha 56. O valor de retorno esperado é falso (*IsFalse*).

Por fim, o quinto caso de teste possui freqüência de 90%, notas 4 e 5 e prova final com valor 7. Nesse caso, o valor de retorno esperado é verdadeiro (*IsTrue*). A classe *Assert* apresenta outros métodos para definição de valores esperados além dos apresentados. Dentre os principais:

- *IsFalse*: Verifica se o parâmetro definido é falso;
- *IsTrue*: Verifica se o parâmetro definido é verdadeiro;
- *AreEqual*: Verifica se os parâmetros são iguais;
- *IsNotNull*: Verifica se o parâmetro não é nulo;
- *IsNull*: Verifica se o parâmetro é nulo;

Executando os Testes

Ao finalizar a programação dos casos de teste, deve-se compilar o projeto para que a DLL seja gerada. Após isso, a mesma deve ser carregada na interface do *NUnit* para que ocorra a execução automatizada dos testes definidos (após a instalação do *NUnit*, o mesmo cria um ícone no *desktop*).

Listagem 2. Classe de Teste

```
1. uses
2.   NUnit.Framework,
3.   untClasses;
4.
5. type
6.   [TestFixture]
7.   TTeste = class
8.     private
9.       objAluno: TAluno;
10.    public
11.      procedure SetUp;
12.      procedure TearDown;
13.      procedure TesteAprovacao1;
14.      procedure TesteAprovacao2;
15.      procedure TesteAprovacao3;
16.      procedure TesteAprovacao4;
17.      procedure TesteAprovacao5;
18.  end;
19.
20.[SETUP]
21.procedure TTeste.SetUp;
22.begin
23.  objAluno := TAluno.Create;
24.end;
25.
26.[TEARDOWN]
27.procedure TTeste.TearDown;
28.begin
29.  objAluno.Free;
30.end;
31.
32.[TEST]
33.procedure TTeste.TesteAprovacao1;
34.begin
35.  objAluno.setDados(60, 0, 0, 0);
36.  Assert.IsFalse(objAluno.CalcularAprovacao, 'Revise o Método');
37.end;
38.
39.[TEST]
40.procedure TTeste.TesteAprovacao2;
41.begin
42.  objAluno.setDados(90, 2, 1, 0);
43.  Assert.IsFalse(objAluno.CalcularAprovacao, 'Revise o Método');
44.end;
45.
46.[TEST]
47.procedure TTeste.TesteAprovacao3;
48.begin
49.  objAluno.setDados(90, 9, 5, 0);
50.  Assert.IsTrue(objAluno.CalcularAprovacao, 'Revise o Método');
51.end;
52.
53.[TEST]
54.procedure TTeste.TesteAprovacao4;
55.begin
56.  objAluno.setDados(90, 4, 5, 4);
57.  Assert.IsFalse(objAluno.CalcularAprovacao, 'Revise o Método');
58.end;
59.
60.[TEST]
61.procedure TTeste.TesteAprovacao5;
62.begin
63.  objAluno.setDados(90, 4, 5, 7);
64.  Assert.IsTrue(objAluno.CalcularAprovacao, 'Revise o Método');
65. end;
```

Abra o *NUnit* e acesse o menu *File>Open*. Após localizar e inserir o arquivo, clique em *Run* para que os testes sejam executados. Ao executar o conjunto dos testes planejados e, se esses passarem com sucesso, a ferramenta apresenta uma barra verde na janela e, em caso contrário, apresenta uma barra vermelha, juntamente com as informações do erro capturado.

A **Figura 4** apresenta o resultado da execução da biblioteca programada no *NUnit*.

Observe que somente o terceiro caso de teste falhou. Isso se deve ao fato que se esperaria que alunos com média 7 fossem aprovados e, na realidade, estão sendo reprovados. De acordo com o *CalcularAprovacao*, quem possuir média entre 3 e 7 (inclusive) deve fazer prova final.

Como esse aluno possui valor 0 na sua prova final, o mesmo foi reprovado, sendo retornado o valor falso, onde o valor esperado deveria ser verdadeiro, falhando assim o caso de teste. Com isso foi possível identificar uma falha no código-fonte, uma vez que o aluno com média de valor 7 deveria ser aprovado.

Deve-se então fazer uma alteração no *CalcularAprovacao* para que sejam aprovados os alunos com média superior ou IGUAL a 7. A alteração deve ser realizada na linha referente a verificação da média ser maior que 7. Modifique-a para "vMedia >= 7".

A **Figura 5** exibe a interface do *NUnit* com os resultados sem falhas após a alteração proposta.

Essa situação caracteriza um defeito no sistema que foi encontrado através de um caso de teste bem planejado, testando os valores limite das condições que, muitas vezes, apresentam problemas na sua programação.

A ferramenta ainda possui uma *interface* via console, onde os testes são visualizados de forma textual. Ideal para ser usada em testes executados automaticamente várias vezes, com a possibilidade de armazenar o resultado em outro arquivo, para futura análise.

Para acessar o *NUnit Console* execute o arquivo *nunit-console.exe* passando como referência a DLL com os testes já compilados. Para armazenar o resultado em algum outro arquivo deve-se utilizar a diretiva ">" do *prompt* do DOS para enviar o resultado para outra saída como, por exemplo, "nunit-console.exe nUnitLibrary.dll > Teste.txt" na linha de comando do DOS.

Conclusão

Uma questão importante numa abordagem como essa, é a definição do número de casos de teste necessários para avaliar todos os caminhos possíveis de um algoritmo. Como é impossível testar todas as situações, torna-se primordial não construir casos de teste desnecessários, testando as mesmas situações.

Uma importante técnica nesse contexto é conhecida como *Complexidade Ciclomática de McCabe*, que avalia o número de caminhos possíveis em um algoritmo, indicando o número mínimo de casos de teste necessários para avaliar todas as suas possibilidades. Entretanto, essa técnica apenas determina o número de casos de teste, não os valores que devem ser utilizados no teste.

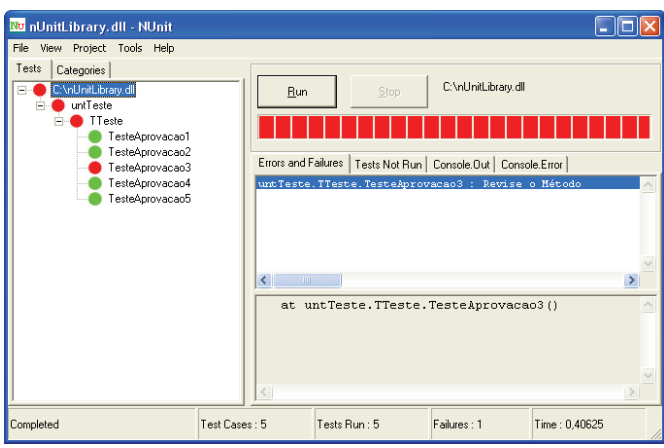


Figura 4. Resultado dos testes no NUnit

Para isso, uma outra técnica conhecida como *Particionamento por Classes de Equivalência*, auxiliada por outra técnica chamada *Análise do Valor Limite*, podem ser úteis, mas fogem ao escopo deste artigo. Entretanto, o sucesso de uma abordagem de desenvolvimento apoiada por testes dá-se em função da qualidade dos casos de teste.

Se um sistema não falha nos testes planejados, fica a dúvida se o sistema é de boa qualidade ou se os casos de teste é que são de baixa qualidade. Portanto, investir num bom planejamento de testes é fundamental para essa estratégia. O ambiente de execução de testes deve ser cuidadosamente planejado.

A execução de testes em dias diferentes, por exemplo, pela simples mudança da data do sistema operacional, pode não representar a mesma situação a ser testada. Assim, é importante garantir que o ambiente seja o mesmo a cada execução do teste, preocupando-se, por exemplo, com o estado das informações no banco de dados. Assim, *Testes de Regressão* são execuções repetidas de um mesmo conjunto de testes, em um mesmo ambiente.

Por fim, além da importância de aplicar testes aos sistemas desenvolvidos para melhorar sua qualidade, independentemente do processo de desenvolvimento, metodologia ou paradigma utilizado, a abordagem de testes automatizados é uma das práticas pregadas pela XP (*eXtreme Programming*) e prática importante na utilização de técnicas de refatoração de código.

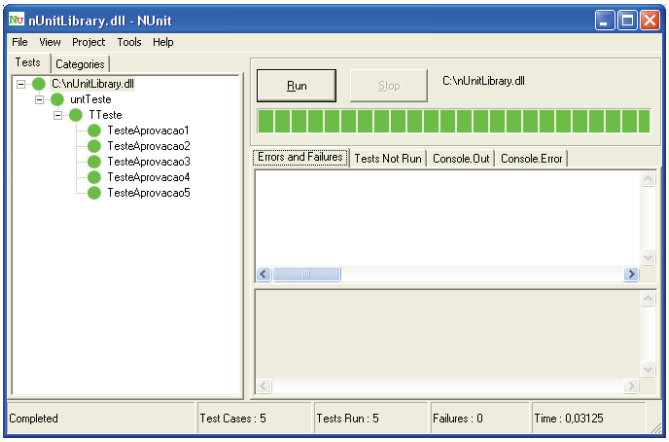


Figura 5. Resultado dos testes no NUnit após a alteração na classe

Assim, a utilização de técnicas de teste proporciona aos desenvolvedores um menor índice de defeitos em seu código-fonte e, conseqüentemente, uma maior confiabilidade do sistema no ambiente de produção.

Nesse sentido, o *NUnit* é um framework de testes unitários para o ambiente de desenvolvimento .NET, que busca automatizar o processo de testes, tornando possível a implantação de uma política de testes na prática. ■

Borland® E-Commerce

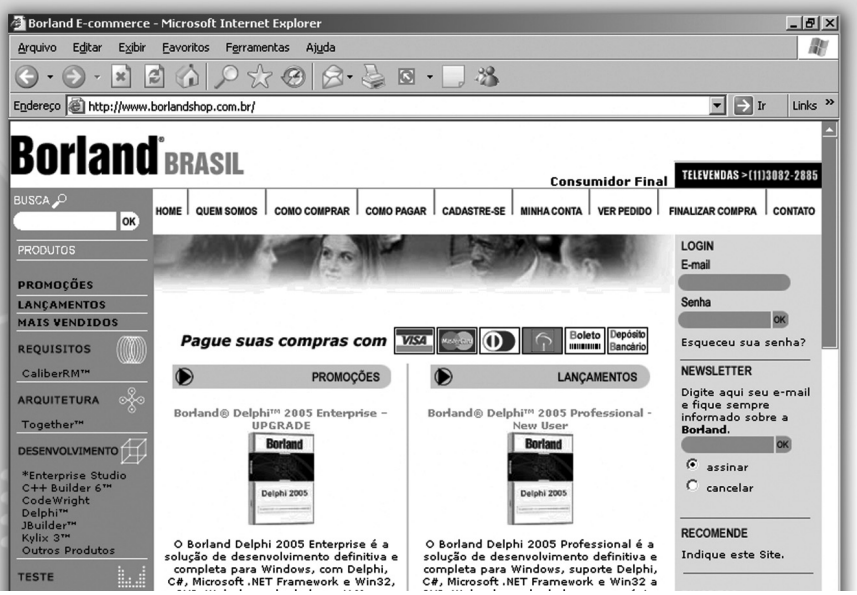
www.borlandshop.com.br

O seu canal direto de compras para produtos Borland

Mais Fácil

Mais Seguro

Mais Econômico



Integrando sua aplicação com a busca de CEPS dos Correios

RODRIGO SENDIN
(rodrigo.sendin@terra.com.br)
é tecnólogo formado pela FATEC de Americana. Trabalha com desenvolvimento de software há mais de 10 anos, escreve artigos para MSDN Magazine e Clube Delphi. Também desenvolve treinamentos de .NET, e atualmente é desenvolvedor na TauNet Consulting.

LUCIANO PIMENTA
(lucianopimenta@clubedelphi.net)
é Técnico em Processamento de Dados, Editor Técnico da Revista ClubeDelphi e Editor do Portal DevMedia (www.devmedia.com.br). Palestrante da 4ª edição da Borland Conference (BorCon).

Você já precisou incluir em suas aplicações uma rotina para busca de CEPs? Se você já desenvolveu algo que envolva cadastros com dados de endereços para entrega, cobrança, faturamento etc., com certeza você já precisou disso.

Uma solução muito comum era usar um arquivo Access que o próprio Correio disponibilizou em seu site ou através de CD, há algum tempo. Nesse banco, tínhamos uma tabela com todos os CEPs do país inteiro, com dados das ruas, bairros, cidades e estados.

Dava um bocado de trabalho, pois tínhamos que manter um banco enorme (como também migrar o banco para a versão que usamos em produção), sem falar das atualizações que eram freqüentes. Hoje, esse banco não é mais disponibilizado e quem utiliza não tem mais atualizações, podendo trabalhar com dados defasados. Então como fazer isso hoje? É exatamente o que veremos neste artigo.

Solução HTML

Se entrarmos no site dos Correios hoje (*www.correios.com.br*), na página principal, temos uma opção na parte inferior chamada *Busca CEP.* Clique sobre ela e veja que temos uma série de opções que envolvem a busca de CEP dos Correios (**Figura 1**).

> **Nota:** As opções do site descritas neste artigo estavam disponíveis até o fechamento dessa edição.

A primeira opção que temos no menu lateral esquerdo, chama-se *Coloque a busca ao CEP no seu site.* Ao clicar nessa opção você verá que basta preencher um pequeno formulário e é possível fazer o download de duas páginas HTML (*busca_cep.html* e *busca_por_cep.html*). O primeiro possui um formulário onde podemos preencher os dados de um endereço e o segundo pergunta apenas o CEP.

Em ambas as páginas o resultado é mostrado em uma nova instância do browser, em uma página dos Correios. Essa é uma solução boa se você quer apenas pesquisar as informações de CEP em seu site.

Mas para alguns, essa pode ser uma solução incompleta, imaginando por exemplo que você queira receber esses dados e colocá-los diretamente em componentes da aplicação, seja ela Web ou Windows.

Consultar o CEP no Office

Outra opção que os Correios oferece é incluir a pesquisa de CEP no Office. Ainda na **Figura 1** veja que a última opção do menu é a *Veja como consultar o CEP no OFFICE da Microsoft*. Clique nessa opção e veja que temos uma explicação para incluir a pesquisa de CEP nas opções de pesquisa do Office.

Seguindo os passos, você registrará um Web Service que permite realizar as pesquisas de CEP. Se você não sabia disso, siga os passos e veja que é uma solução bem interessante, principalmente se você ou seus usuários usam muito o Office.

O Web Service

Apesar de não estar claramente divulgado no site dos Correios, a busca de CEP no Office usa um Web Service, que naturalmente pode ser utilizado em nossas aplicações .NET. O endereço é *http://consultacep.correios.com.br/office2003/Query.asmx*. Devo adverti-los que essa não é uma tarefa muito simples.

Esse Web Service foi desenvolvido para ser consumido pelo Office, portanto temos um certo esforço para utilizá-lo em aplicações .NET. Vamos começar com um teste simples. Abra o Delphi for .NET (versão 8, 2005 ou 2006) e crie uma nova aplicação ASP.NET (você pode fazer em Windows Forms – Desktop se quiser). Dê o nome de "CEP" para a aplicação. Na página *Default.aspx* crie uma interface igual a demonstrada na **Figura 2**.

Veja que esse é um formulário bem simples onde estamos solicitando o CEP em um *TextBox* chamado "txtCEP". Abaixo temos um botão chamado "btnPesquisar" e um outro *TextBox* chamado "txtRetorno", onde apresentaremos o retorno do Web Service dos Correios.

No *Project Manager* clique com o botão direito sobre o projeto. Escolha a opção *Add Web Reference*, informe no endereço de pesquisa o Web Service dos Correios (*http://consultacep.correios.com.br/office2003/Query.asmx*) e clique no botão *Go* (uma seta azul).

Veja na **Figura 3** que o Web Service dos Correios expõe uma

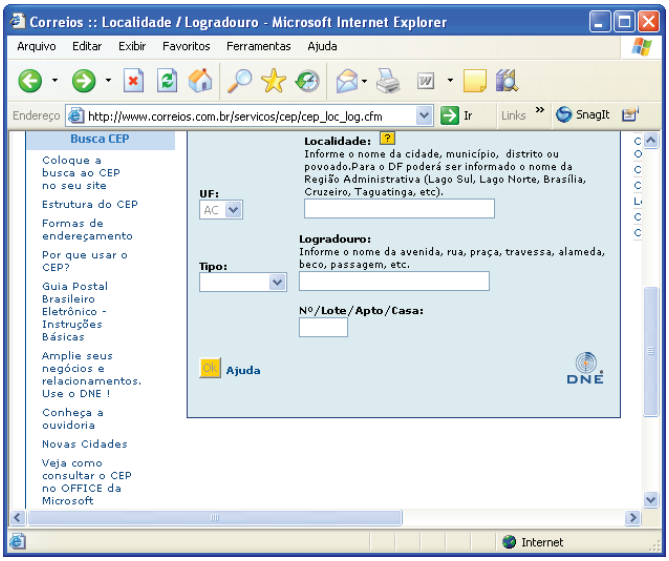


Figura 1. Opções de busca de CEP disponíveis no site dos Correios

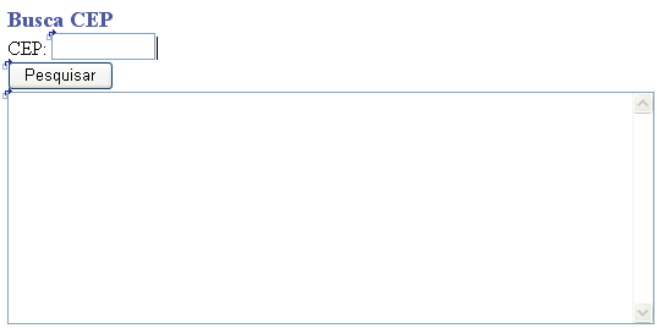


Figura 2. Interface para a busca de CEP

classe chamada *QueryProcessor*, que possui um método chamado *Query*. Para adicionar a referência, basta clicar em *Add Reference*.

Para continuarmos, vá ao código da *Default.aspx* e inclua no *uses* os seguintes namespaces:

```
uses System.IO, System.Xml, System.Text,
  correios.Query;
```

Em seguida, ainda no código da página *Default.aspx*, crie uma função, na seção *private*, chamada *GeraXML*. Adicione o código da **Listagem 1** na referida função.

Como você deve estar imaginando, esse método gerará um XML contendo o CEP que será informado no *txtCEP*. Isso é necessário, pois o Web Service foi feito para receber pesquisas de CEP do Office, portanto ele recebe um XML formatado nesses padrões.

Veja que através de um *MemoryStream* estamos criando um *XmlTextWriter* em memória, que no final é retornado como *string* por um *StreamReader*. A única informação que esse XML carrega é o CEP informado no *TextBox*.

Para finalizar, volte ao *design* da página, dê um duplo clique no botão e inclua o código da **Listagem 2**.

ClubeDelphi PLUS

Acesse agora mesmo o Portal do Assinante ClubeDelphi e assista a uma vídeo-aula de Luciano Pimenta que mostra como criar e consumir Web Services no Delphi 2006.

www.devmedia.com.br/articles/viewcomp.asp?comp=2878

Listagem 1. Código que gera o XML para a busca de CEP

```
function GeraXML: string;
var
  Stream: MemoryStream;
  XmlWriter: XmlTextWriter;
  Reader: StreamReader;
begin
  Stream := MemoryStream.Create;
  XmlWriter := XmlTextWriter.Create(
    Stream, Encoding.UTF8);
  XmlWriter.WriteStartDocument();
  XmlWriter.WriteStartElement('QueryPacket',
    'urn:Microsoft.Search.Query');
  XmlWriter.WriteStartElement('Query');
  XmlWriter.WriteStartElement('Context');
  XmlWriter.WriteStartElement('Context');
  XmlWriter.WriteStartElement('QueryText');
  XmlWriter.WriteString(txtCEP.Text);
  XmlWriter.WriteEndElement();
  XmlWriter.WriteEndElement();
  XmlWriter.WriteStartElement('OfficeContext',
    'urn:Microsoft.Search.Query.Office.Context');
  XmlWriter.WriteStartElement('ApplicationContext');
  XmlWriter.WriteStartElement('Name');
  XmlWriter.WriteString('Microsoft Office');
  XmlWriter.WriteEndElement();
  XmlWriter.WriteStartElement('Version');
  XmlWriter.WriteString('(11.0.6568)');
  XmlWriter.WriteEndElement();
  XmlWriter.WriteEndElement();
  XmlWriter.WriteEndElement();
  XmlWriter.WriteEndElement();
  XmlWriter.WriteEndElement();
  XmlWriter.WriteEndElement();
  XmlWriter.WriteEndDocument();

  XmlWriter.Flush();
  Stream.Flush();
  Stream.Position := 0;
  Reader := StreamReader.Create(Stream);
  XmlWriter := nil;
  Stream := nil;
  Result := Reader.ReadToEnd().ToString();
end;
```

Listagem 2. Evento Click que fará a chamada do Web Service

```
var
  Busca: correios.Query.QueryProcessor;
begin
  txtRetorno.Text := '';
  try
    Busca := correios.Query.QueryProcessor.Create;
    txtRetorno.Text := Busca.Query(GeraXML);
  except
    on E: Exception do
    txtRetorno.Text := E.ToString;
  end;
end;
```

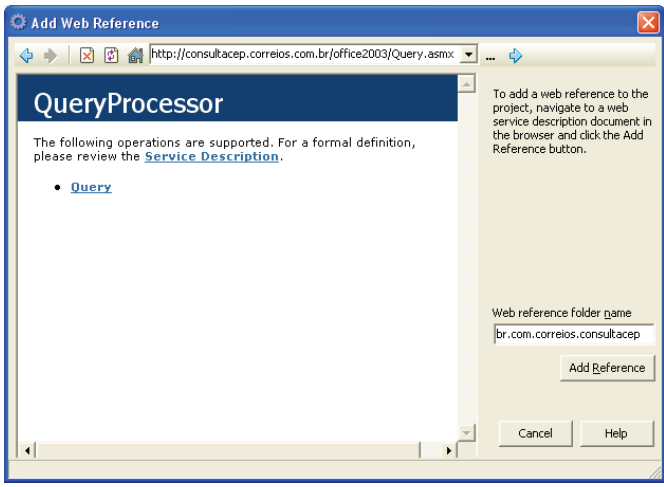


Figura 3. Adicionando a referência ao Web Service

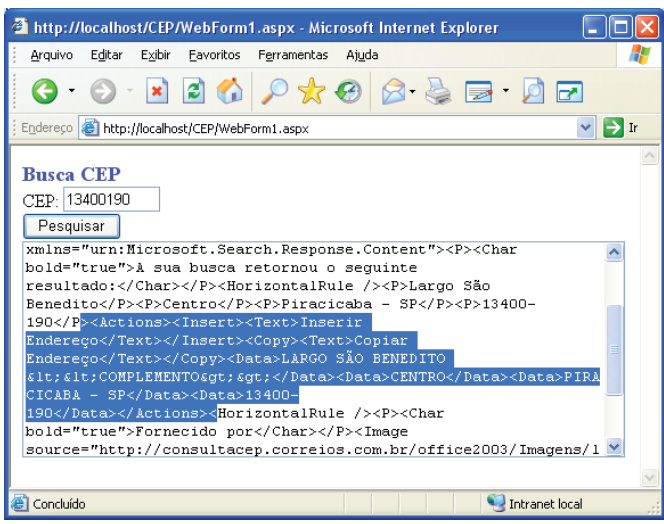


Figura 4. Testando a busca de CEP simples

Observe que estamos fazendo a chamada ao Web Service utilizando o *Query*, passando como parâmetro a *string* que contém o XML que criamos no *GeraXML*. Vamos fazer um teste? Salve, compile e execute o projeto.

Como mostra a **Figura 4**, informe o CEP sem nenhum caractere de máscara e clique em *Pesquisar*.

Veja que o retorno do Web Service também é um XML e que se você observar o conteúdo desse, verá que dentro da *Tag* *Actions*, estão os dados de Endereço do CEP em questão. Basta então, recuperar essas informações utilizá-las da forma mais conveniente para a aplicação.

Busca avançada

Vimos como realizar uma busca simples, onde através do CEP podemos pesquisar qual é o endereço. Agora vamos fazer uma busca avançada, onde, através do endereço podemos pesquisar qual é o CEP.

Para isso, vamos criar uma nova página (você pode usar a mesma página, mas por questões de organização, vamos criar outra). Inclua alguns componentes, como mostra a **Figura 5** (adicionei uma tabela para organizar melhor o alinhamento dos componentes).

Veja que esse é um formulário onde preenchemos todos os dados de endereço. Utilizamos apenas *TextBoxes*, e cada um deles foi nomeado de acordo com o seu conteúdo: "txtEstado", "txtCidade", "txtLogradouro", "txtEndereco", "txtNumero" e "txtCep".

ClubeDelphi PLUS

Acesse agora mesmo o Portal do Assinante ClubeDelphi e assista a uma vídeo-aula de Guinther Pauli que mostra como criar arquivos XML com o XmlTextWriter.

www.devmedia.com.br/articles/viewcomp.asp?comp=3275

O *TextBox* que receberá o XML tem o mesmo nome do anterior (*txtRetorno*), assim como o botão *Pesquisar* (*btnPesquisar*). Em seguida, adicione mais uma função, chamada *GeraXMLAvancado* e codifique-a com o código da **Listagem 3**.

Listagem 3. Função que gera o XML para a busca avançada

```
uses System.IO, System.Xml, System.Text,
  correios.Query;
...
function GeraXMLAvancado: string;
var
  Stream: MemoryStream;
  XmlWriter: XmlTextWriter;
  Reader: StreamReader;
begin
   Stream := MemoryStream.Create();
   XmlWriter := XmlTextWriter.Create(Stream, Encoding.UTF8);
   XmlWriter.WriteStartDocument();
   XmlWriter.WriteStartElement('QueryPacket',
     'urn:Microsoft.Search.Query');
   XmlWriter.WriteStartElement('Query');
   XmlWriter.WriteStartElement('Context');
   XmlWriter.WriteStartElement('QueryText');
   XmlWriter.WriteString('Logradouro');
   XmlWriter.WriteEndElement();
   XmlWriter.WriteStartElement('Requery');
  XmlWriter.WriteStartElement('ServiceParameters',
    'urn:Microsoft.Search.Office.ServiceParameters');
   XmlWriter.WriteStartElement('Parameters');
   XmlWriter.WriteStartElement('AdvancedSearchEditUF');
   XmlWriter.WriteString(txtEstado.Text);
   XmlWriter.WriteEndElement();
   XmlWriter.WriteStartElement('AdvancedSearchEditLocalidade');
   XmlWriter.WriteString(txtCidade.Text);
   XmlWriter.WriteEndElement();
   XmlWriter.WriteStartElement('AdvancedSearchEditTipoLogradouro');
   XmlWriter.WriteString(txtLogradouro.Text);
   XmlWriter.WriteEndElement();
   XmlWriter.WriteStartElement('AdvancedSearchEditLogradouro');
   XmlWriter.WriteString(txtEndereco.Text);
   XmlWriter.WriteEndElement();
   XmlWriter.WriteStartElement('AdvancedSearchEditNumero');
   XmlWriter.WriteString(txtNumero.Text);
   XmlWriter.WriteEndElement();
   XmlWriter.WriteEndElement();
   XmlWriter.WriteEndElement();
   XmlWriter.WriteEndElement();
   XmlWriter.WriteEndElement();
   XmlWriter.WriteStartElement('OfficeContext',
     'urn:Microsoft.Search.Query.Office.Context');
   XmlWriter.WriteStartElement('ApplicationContext');
   XmlWriter.WriteStartElement('Name');
   XmlWriter.WriteString('Microsoft Office');
   XmlWriter.WriteEndElement();
   XmlWriter.WriteStartElement('Version');
   XmlWriter.WriteString('(11.0.6568)');
   XmlWriter.WriteEndElement();
   XmlWriter.WriteEndElement();
   XmlWriter.WriteEndElement();
   XmlWriter.WriteEndElement();
   XmlWriter.WriteEndElement();
   XmlWriter.WriteEndDocument();
   XmlWriter.Flush;
   Stream.Flush;
   Stream.Position := 0;
   Reader := StreamReader.Create(Stream);
   XmlWriter := nil;
   Stream := nil;
   Result := Reader.ReadToEnd.ToString;
end;
```

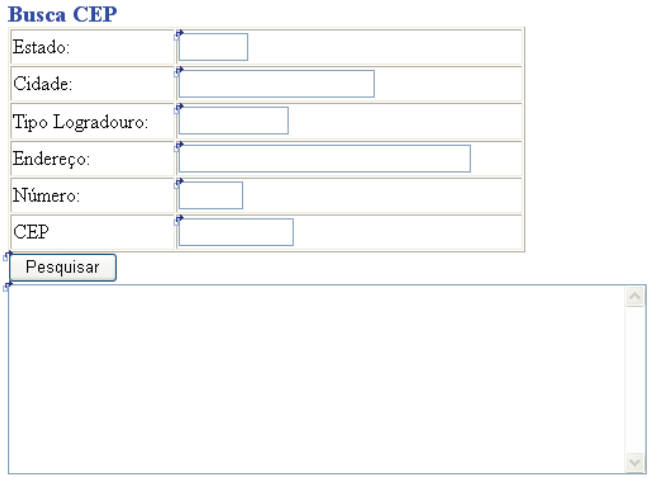


Figura 5. Formulário para busca de CEP avançada

Como você pode observar, o XML gerado é bem parecido com o anterior, seguindo os mesmos padrões. A grande diferença é que nesse, estamos passando os dados de um Endereço para que o Web Service retorne o CEP ou os CEPs do mesmo.

Como você poderá ver na prática, é possível realizar a pesquisa com ou sem o número do logradouro. Caso seja feita uma pesquisa sem o número, é provável que mais de um CEP seja encontrado. Vamos fazer um teste, adicione no *btnPesquisar* o código da **Listagem 4**.

Listagem 4. Realiza a busca avançada

```
var
  Busca: correios.Query.QueryProcessor;
begin
  txtRetorno.Text := '';
  try
    Busca := correios.Query.QueryProcessor.Create;
    txtRetorno.Text := Busca.Query(GeraXMLAvancado);
  except
    on E: Exception do
    txtRetorno.Text := E.ToString;
  end;
end;
```

Salve, compile e execute (seja você criou uma nova página, verifique se a mesma é a página inicial da aplicação). Como mostra a **Figura 6**, preencha os campos: *Estado*, *Cidade*, *Tipo Logradouro* e *Endereço* e em seguida clique em *Pesquisar*.

Veja no *TextBox* de retorno que o XML é bem maior e mais complexo que o anterior, na busca simples. Nesse XML temos

+ de 80.000 membros cadastrados
+ de 15.000 exemplos com fontes
+ de 900 apostilas
+ de 4.000 dicas
Fórum Delphi
Artigos
TOTALMENTE GRÁTIS
www.delphi.eti.br
Um dos maiores sites de apoio a desenvolvedores Delphi do Brasil!!!

primeiramente uma orientação do próprio Web Service, dizendo que os campos de *Estado* e *Tipo Logradouro* precisam respeitar um conjunto de valores válidos, informados no XML. Isso ajuda a preencher nossas próprias listas de seleção para que o usuário possa escolher os valores.

E veja que o resultado da pesquisa é retornado dentro da *tag Heading*. Faça outros testes e veja que alguns endereços poderão retornar mais que um CEP identificados também na *tag Heading*.

Recuperando o resultado em componentes

Vamos agora criar uma rotina para recuperar os valores retornados pelo Web Service no XML e incluir esses valores em componentes da nossa página. Faremos um exemplo com a consulta simples de CEP, para isso sua página deve ficar como demonstrada na **Figura 7** (crie uma nova página na aplicação).

Veja que nesse formulário apenas perguntaremos o CEP (no "txtCEP") e colocaremos o resultado da pesquisa em quatro *TextBoxes*: "txtEndereco", "txtBairro", "txtCidade" e "txtEstado". Assim podemos simular o preenchimento automático desses campos em um formulário de cadastro de clientes.

Adicione na página criada o *GeraXML* do primeiro exemplo. Não esqueça de adicionar no *uses* os namespaces mostrados anteriormente, juntamente com *System.Collections* para que possamos trabalhar com um *ArrayList*. Em seguida, inclua o código da **Listagem 5** no evento *Click* do *btnPesquisar*.

Listagem 5. Recupera valores do XML e coloca em TextBoxes

```
var
  Busca: correios.Query.QueryProcessor;
  retorno: string;
  XmlDoc: XmlDocument;
  campos: ArrayList;
  CidadeUF: &Array;
begin
  try
    Busca := correios.Query.QueryProcessor.Create;
    retorno := Busca.Query(GeraXML);
    XmlDoc := XmlDocument.Create;
    XmlDoc.LoadXml(retorno);
    campos := ArrayList.Create;
    campos := VarreXML(XmlDoc.DocumentElement.
      ChildNodes, campos);
    { Armazenando Endereço e Bairro }
    txtEndereco.Text := campos[0].ToString.Replace(
      '<<COMPLEMENTO>>', '');
    txtBairro.Text := campos[1].ToString;
    { Separando Cidade do Estado }
    CidadeUF := campos[2].ToString.Split(['-']);
    txtCidade.Text := CidadeUF[0].ToString.Trim;
    txtEstado.Text := CidadeUF[1].ToString.Trim;
  except
    on E: Exception do
    Response.Write(E.ToString);
  end;
end;
```

Observações da Listagem 5

Veja que através do Replace estamos removendo a string "<<COMPLEMENTO>>" que vem no Endereço no XML. Através do Split separamos a Cidade do Estado, que no XML vem no mesmo campo.

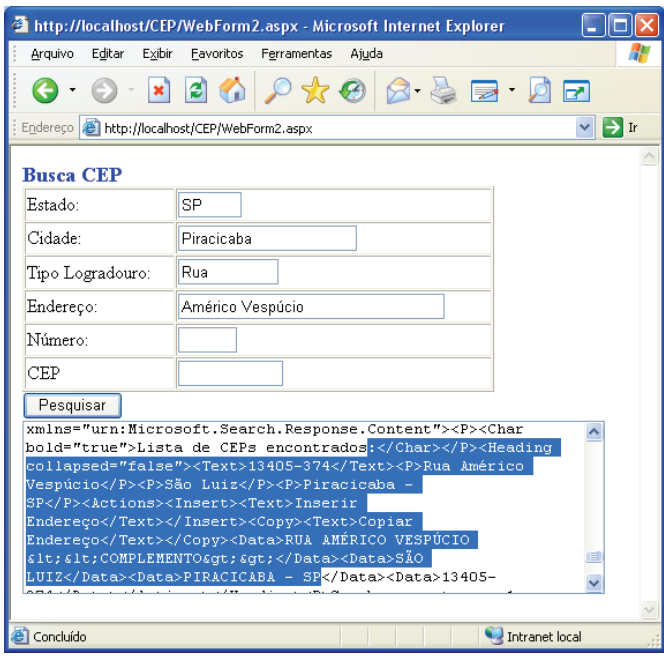


Figura 6. Resultado da busca avançada de CEP

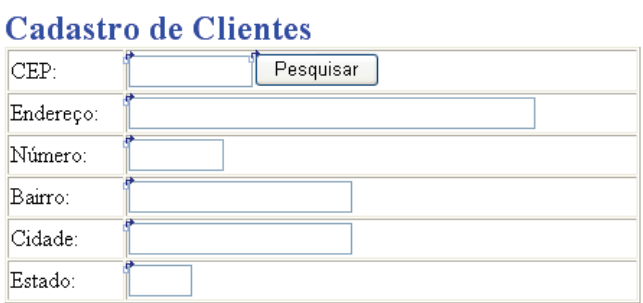


Figura 7. Formulário para retorno do resultado em componentes

Veja que estamos fazendo a chamada ao *GeraXML* que criamos anteriormente para a busca de CEP simples, e o retorno está sendo armazenado em uma *string*. Através de um objeto da classe *XmlDocument* estamos carregando a *string* com o conteúdo do XML.

Existem diversas formas de ler um XML, estamos utilizando o *XmlDocument* por ser uma das mais simples, porém você pode utilizar a classe que preferir.

Observe que após carregar o *XmlDocument* estamos criando um *ArrayList* chamado *campos*. Em seguida estamos chamando o *VarreXML*, passando como parâmetro os *ChildNodes* do *XmlDocument* e a *ArrayList* que criamos.

O retorno dessa função é a própria *ArrayList*, que conterá os valores para preenchermos os *TextBoxes*. Crie o *VarreXML*, conforme a **Listagem 6**.

Veja que através de um *for in*, percorremos todos os *XmlNodes* da coleção *nodes*. E caso o *node* tenha o nome *Data* estamos adicionando o seu valor no *ArrayList*. Observe que cada *XmlNode* de um *XmlDocument* pode possuir mais um conjunto de *nodes*, que podem ser acessados pela propriedade *ChildNodes*.

Listagem 6. Percorre todo o XML em busca dos Valores

```
function VarreXML(nodes: XmlNodeList;
  campos: ArrayList): ArrayList;
var
  node: XmlNode;
begin
   for node in nodes do
   begin
     if (node.Name.Trim = 'Data') then
       campos.Add(node.InnerText);
     VarreXml(node.ChildNodes, campos);
  end;
  Result := campos;
end;
```

Dessa forma, para cada *XmlNode* estamos novamente chamando o *VarreXml*, e assim varreremos todo o conteúdo do XML de forma recursiva. Ao final a função, devolverá o *ArrayList* com os valores encontrados.

Vamos fazer um teste? Salve, compile e execute o projeto. Como mostra a **Figura 8**, informe um CEP válido e clique em *Pesquisar*.

Veja que os *TextBoxes* serão preenchidos com os valores retornados pelo Web Service, bastando que o usuário preencha o campo número. Essa é uma ótima solução em telas de cadastro com endereços, imagine o tempo que ganhamos ao ter esses dados preenchidos automaticamente pelo Web Service.

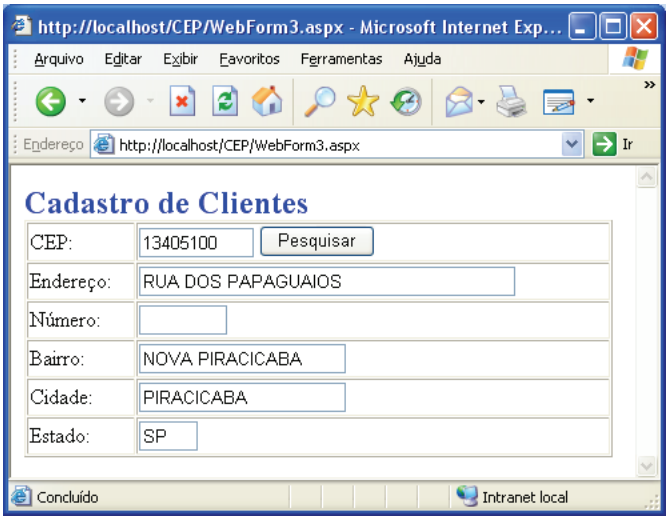


Figura 8. Testando a pesquisa de CEP tratada

Recuperando o resultado em um DataGrid

Como sabemos, existem ruas que possuem mais de um CEP, para esses casos, quando fizermos uma pesquisa avançada o Web Service retornará uma lista de todos os CEPs possíveis do endereço em questão.

Para finalizarmos o artigo, vamos fazer um último exemplo onde o usuário informará um Endereço, e os CEPs retornados serão apresentados em um *DataGrid*. Para isso, crie uma nova página e adicione componentes para que fique igual à da **Figura 9**.

Veja que temos todos os campos para a busca de CEP avançada (adicione a função *GeraXMLAvancado* nessa página e dê aos controles os nomes que utilizamos nos exemplos anteriores), e que incluímos um *DataGrid* na página.

Nesse *DataGrid* incluímos apenas uma coluna *Select* do tipo *Button Column* (acesse o editor do *DataGrid* e adicione a coluna), assim quando o usuário clicar em *Select* o CEP escolhido será mostrado no *TextBox* (mostraremos esse código a seguir).

Ainda no *DataGrid*, é preciso manter a propriedade *AutoGenerateColumns* igual a *True*. Dê um duplo clique no botão *Pesquisar* e inclua o código da **Listagem 7**.

Veja que o código é bem parecido com o do exemplo anterior, com a diferença que estamos chamando a pesquisa de CEP avançada e que ao invés de recuperarmos os dados em um *ArrayList*, estamos usando um *DataTable*.

Veja que criamos quatro colunas no *DataTable*: *CEP*, *Logradouro*, *Bairro* e *CidadeUF*. Para preenchermos o *DataTable* com os resultados do XML, estamos utilizando o *VarreXMLAvancado*, que é um pouco diferente do que foi utilizado no exemplo anterior, pois retorna um *DataTable*. Confira como ficou o código na **Listagem 8**.

Nesse código, estamos novamente varrendo o XML de forma recursiva, só que dessa vez procurando pelo node *Heading*. Cada resultado retornado pela pesquisa virá dentro desse node. Para finalizar, basta codificarmos o evento *SelectedIndexChanged* do *DataGrid*.

Esse evento é disparado toda vez que clicarmos no *Select* de qualquer uma das linhas. Para codificá-lo, acesse o evento e adicione o seguinte código:

```
txtCEP.Text := DataGrid1.SelectedItem.Cells[1].Text;
```

Vamos testar? Salve, compile e execute seu projeto. No formulário informe um endereço que contenha mais que um CEP, e clique em *Pesquisar*. Veja que no *DataGrid* são apresentados todos os CEPs encontrados para esse endereço, assim como mostra a **Figura 10**. Agora, basta você clicar no botão *Select* de qualquer uma das linhas que o CEP correspondente será preenchido no *TextBox*.

Conclusão

Como você pôde conferir neste artigo, é possível realizar buscas de CEP no Web Service que os Correios disponibilizam para o Office da Microsoft. Infelizmente esse Web Service está formatado para receber e retornar um XML nos padrões da ferramenta de pesquisa do Office, mas nada que nos impeça de utilizá-lo, respeitando esses padrões. Veja que as possibilidades de tratamento são muitas, basta você avaliar as suas necessidades de pesquisa e por a mão na massa. Daqui para frente é com você!

Não perca na próxima edição, onde mostraremos como criar uma consulta ao mesmo Web Service a partir de uma aplicação Delphi 7 (VCL Win32). Até lá! ■

Links

Site dos Correios

www.correios.com.br

Listagem 7. Evento para a busca de CEP

```
uses System.IO, System.Xml, System.Text,
  correios.Query;
...
var
  Busca: correios.Query.QueryProcessor;
  retorno: string;
  XmlDoc: XmlDocument;
  Ceps: DataTable;
begin
  try
    Busca := correios.Query.QueryProcessor.Create;
    retorno := Busca.Query(GeraXMLAvancado);
    XmlDoc := XmlDocument.Create;
    XmlDoc.LoadXml(retorno);
    Ceps := DataTable.Create;
    Ceps.Columns.Add('CEP');
    Ceps.Columns.Add('Logradouro');
    Ceps.Columns.Add('Bairro');
    Ceps.Columns.Add('CidadeUF');
    Ceps := VarreXmlAvancado(
      XmlDoc.DocumentElement.ChildNodes, Ceps);
    DataGrid1.DataSource := Ceps;
    DataGrid1.DataBind;
  except
    on E: Exception do
    Response.Write(E.ToString);
  end;
end;
```

Listagem 8. Percorre todo o XML e preenche o DataTable

```
function VarreXMLAvancado(
  nodes: XmlNodeList; Ceps: DataTable): DataTable;
var
  node: XmlNode;
  Row: DataRow;
begin
  for node in nodes do
  begin
    if (node.Name.Trim = 'Heading') then
    begin
      Row := Ceps.NewRow;
      Row['CEP'] := node.ChildNodes[0].InnerText;
      Row['Logradouro'] :=
        node.ChildNodes[1].InnerText;
      Row['Bairro'] := node.ChildNodes[2].InnerText;
      Row['CidadeUF'] := node.ChildNodes[3].InnerText;
      Ceps.Rows.Add(Row);
    end;
    VarreXmlAvancado(node.ChildNodes, Ceps);
  end;
   Result := Ceps;
end;
```

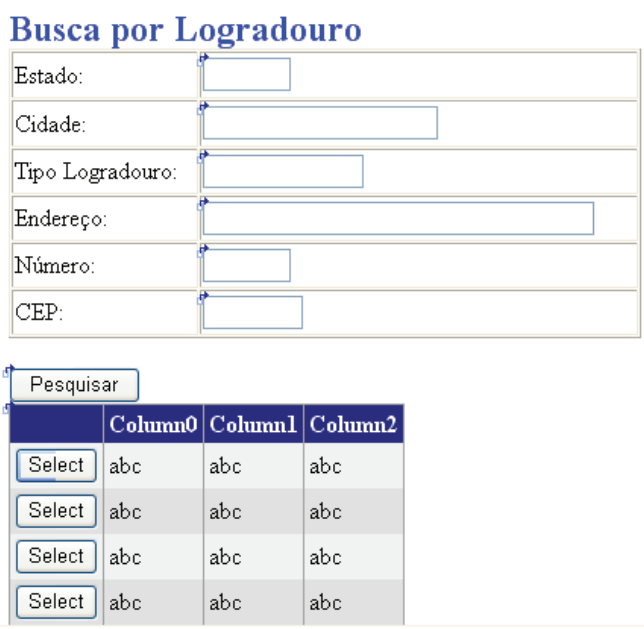


Figura 9. Pesquisa de CEP em um DataGrid

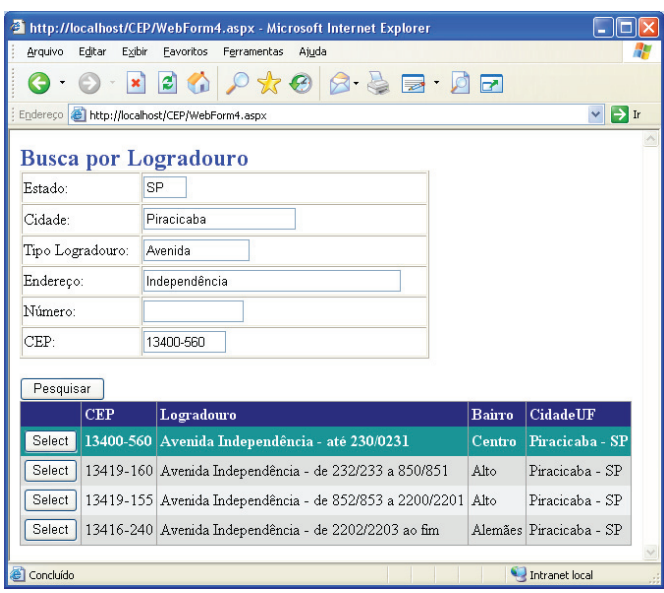


Figura 10. Teste da pesquisa de CEP em um DataGrid

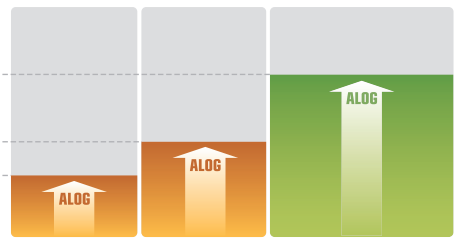
FAÇA COMO A ALOG
FOQUE EM SEU NEGÓCIO

A ALOG é totalmente focada em serviços de Data Center. Dedica-se 24 horas por dia, 7 dias por semana a cuidar do negócio dos nossos 320 clientes corporativos.

ALOG desenvolveu no Brasil o conceito de **HOSTING GERENCIADO**: um serviço de excelência que garantirá a sua empresa estabilidade, agilidade, redundância, e escalabilidade. Para que isso seja verdade os clientes da ALOG contam com **Suporte Incondicional***, um conceito revolucionário em atendimento no Brasil.

ENTENDA O **HOSTING GERENCIADO**

CAMADAS DE NEGÓCIO	DESCRIÇÃO
Processo de Negócio Aplicação do Cliente	Atividades cotidianas do cliente Softwares de Gestão (Ex: ERP, CRM, outros)
Ambiente da Aplicação Sistema Operacional	.NET, JAVA, SQL Server, MySQL, Oracle, Exchange Windows, Linux, Unix, Solaris
Servidor ou equipamento	Servidores RISC/CISC, Appliances, Roteadores, Switches
Rede Infra-estrutura Básica	IP, Wan, Rede Local Energia, HVAC, Segurança - 2N+1



ESTRATÉGIA DO CLIENTE	COLOCATION	HOSTING	HOSTING GERENCIADO

PLANOS DE **HOSTING GERENCIADO**

GERENCIADO 1

- **Pentium D 2.8 GHz**
- 512 RAM DDR2
- 2 HD SATA 80 GB
- 450 GB de Transf. Mensal
- Firewall e Backup

BOOT REMOTO
GERENCIAMENTO**

R$1.050,00 mensais
INSTALAÇÃO IMEDIATA***

GERENCIADO 2

- **Dual Xeon Dual Core 3.0 GHz**
- 1GB RAM DDR2 533
- 2 HD SATA RAID I 300 GB
- 1.000 GB de Transf. Mensal
- Firewall e Backup

BOOT REMOTO
GERENCIAMENTO**

R$2.850,00 mensais
INSTALAÇÃO IMEDIATA***

ALOG data centers

Excelência em Hosting
www.alog.com.br :: 0800 282 3330

Hosting Gerenciado | Colocation | Contingência | Email Corporativo | Conectividade | Serviços Profissionais

Conheça na prática alguns componentes da fantástica JVCL

MICHAEL BENFORD
(michael@devmedia.com.br)
é acadêmico do curso de Matemática da Universidade Federal Fluminense e desenvolvedor Delphi e .NET da QSi – Qualidade em Sistemas de Informação. Programa em Delphi há 5 anos, desenvolvendo aplicações cliente/servidor, multicamadas, Web, utilitários de uso geral, componentes, experts e aplicativos não-comerciais. É colunista do Portal DevMedia, articulista e membro da comissão editorial da revista ClubeDelphi. Palestrante da primeira edição do Developer's Webdays e da BorCon Brasil 2006.

Componentes são coisas onipresentes em aplicações desktop. É raro encontrar um desenvolvedor que abra mão deles na tentativa de ser mais produtivo, pois afinal, a razão da sua existência é a produtividade que eles proporcionam.

O Delphi possui centenas de componentes e controles, agrupados dentro da VCL, que permitem a criação de sofisticadas interfaces com o usuário, conhecidas como GUI's (*Graphical User Interface*), bem como a implementação de códigos complexos. Porém, em determinadas situações, sentimos a falta de um ou outro componente/controle para realizar uma tarefa mais específica.

Nessas horas, temos dois caminhos a seguir: criamos por conta própria, "no braço" ou recorremos a terceiros, que já o fizeram por nós. Existem dezenas de suítes de componentes disponíveis na Internet, algumas gratuitas, outras pagas.

Dentre todas elas, com certeza, a que tem recebido maior destaque, pela sua qualidade e pelo fato de ser *open source*, é a JVCL, acrônimo de *JEDI Visual Component Library*. Neste artigo vamos conhecer 20 componentes selecionados de sua vasta biblioteca de mais de 600, que podem fazer a diferença em nossas aplicações.

Um breve comentário sobre o JEDI

O nome JEDI foi inspirado na trilogia *Star Wars* e significa *Joint Endeavour of Delphi Innovators*, ou, em uma tradução livre, *Esforço Conjunto de Inovadores Delphi*. Como o nome diz, trate-se de uma organização de desenvolvedores de todo o mundo, com o objetivo de prover aos programadores Delphi acesso rápido à API's, bibliotecas e qualquer outro recurso que estenda a capacidade do IDE da ferramenta.

Tudo isso de forma gratuita, no formato *open source*. Como resultado desse esforço, temos:

- *JVCL*: extensa suíte de componentes;
- *JCL*: vasta biblioteca de rotinas para os mais variados fins;

- *JVCS*: excelente aplicativo para versionamento de arquivos;
- *JCodeFormatter*: ferramenta para formatar automaticamente o código-fonte, seguindo regras de padronização.

Existem muitos outros projetos sob a administração do grupo e você pode conferi-los no site oficial: *www.delphi-jedi.org*. Sem mais rodeios, vamos direto ao que interessa.

Dica: Para saber mais sobre o JEDI VCS, consulte a edição 73 da ClubeDelphi.

Baixando a JVCL

Acesse o site *jvcl.sourceforge.net* e clique em *Downloads*, no menu à esquerda. Na próxima página, clique em *JVCL 3*, na coluna *Package*. Em seguida, baixe o arquivo *JVCL320CompleteJCL197-Build2172.*zip, que tem aproximadamente 18 MB.

Se o leitor observar bem o nome do arquivo, vai notar que é necessário baixar também a JCL, pois a JVCL a utiliza extensivamente.

Nota: Até a data de fechamento dessa edição, a JVCL encontrava-se na versão 3.20 e a JCL na versão 1.97.

Após realizar o download do arquivo, descompacte-o para uma pasta de fácil acesso, como por exemplo, "C:\JEDI". Esse processo deve demorar alguns minutos, já que são milhares de arquivos contidos no pacote.

Como mencionado anteriormente, a JVCL precisa da JCL, e, portanto, essa deve ser instalada antes. Abra o diretório escolhido, na pasta *jcl*, execute o arquivo *install.bat*. O instalador será construído *on the fly* e executado em seguida (**Figura 1**).

Note que há uma aba para cada versão instalada do Delphi, contendo as opções específicas de cada uma. Para os exemplos empregados neste artigo, deixe-as como estão e clique no botão *Install*. O instalador compilará os pacotes necessários para cada versão do Delphi e o processo pode demorar alguns minutos para terminar.

Após instalar a JCL, vamos agora fazer o mesmo para a JVCL. Volte novamente à pasta onde você descompactou o arquivo ZIP, procure pelo diretório *jvcl* e execute mais uma vez o arquivo *install.bat*. Isso irá produzir o instalador da JVCL (**Figura 2**).

Marque a caixa *New installation/upgrade from an older version* e clique em *Next*. Na próxima tela, escolha em quais versões do Delphi você deseja instalar o pacote e clique em *Next* novamente. Nas telas seguintes, deixe as opções como estão e avance até o botão mudar de nome para *Install*.

Clique nele e aguarde o processo de compilação dos pacotes, que, assim como na JCL, pode levar algum tempo para ser concluído. Alguns *warnings* podem ser gerados, mas isso é esperado. Clique em *Finish* para finalizar a instalação.

Pronto! Agora podemos começar a explorar os componentes da JVCL.

Nota: Em alguns casos é necessário adicionar manualmente diretórios no *Library Path* do Delphi, como *jcl\source\common* e *jcl\source\windows*.

Mensagens pop-ups estilo Outlook

O *JvDesktopAlert* (paleta *JvDialogs*) permite criar mensagens *popup*, iguais ao do Outlook (versão *full*) exibe quando chega um novo e-mail. Para exemplificar o seu comportamento, crie uma nova aplicação e adicione o componente ao formulário.

ClubeDelphi PLUS

Acesse agora mesmo o Portal do Assinante ClubeDelphi e assista a uma vídeo-aula de Luciano Pimenta que mostra como baixar e instalar o JEDI no Delphi.

www.devmedia.com.br/articles/viewcomp.asp?comp=3655

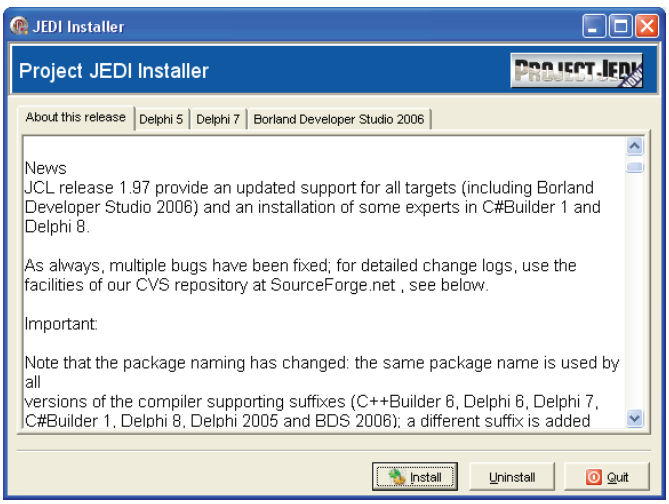


Figura 1. Instalador da JCL

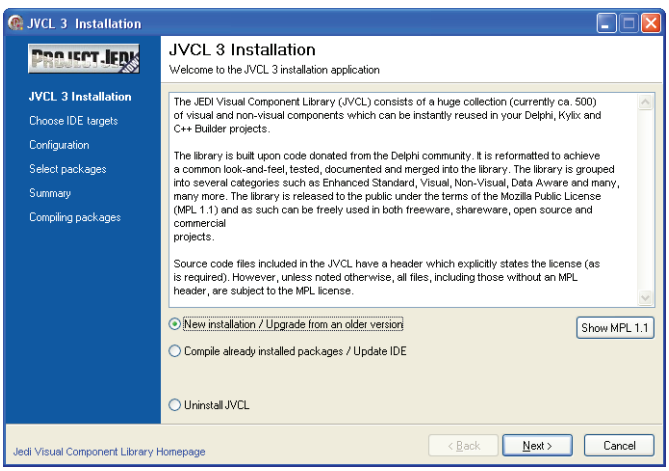


Figura 2. Instalador da JVCL

Na propriedade *Image*, adicione uma imagem de sua preferência. Inclua também um *ImageList* (*Win32*), adicione algumas imagens e aponte-o para a propriedade *Images* do *JvDesktopAlert*.

Em seguida, dê um duplo clique na propriedade *Buttons* e adicione alguns itens, configurando a propriedade *ImageIndex* de cada um para um índice menor ou igual à quantidade de imagens que você colocou no *ImageList*.

Por fim, insira um botão e implemente seu evento *OnClick* como mostrado na **Listagem 1**.

Compile a aplicação e clique no botão. Você verá um alerta no canto inferior direito da sua tela (**Figura 3**).

Algumas propriedades do componente que você deve conhecer são:

- *AlertStyle*: configura o efeito utilizado na exibição do alerta;
- *Colors*: define as cores que o alerta será pintado;
- *Location*: indica a posição da tela em que o alerta aparecerá;
- *Options*: configura se o alerta pode ser clicado, movido ou fechado;
- *StyleOptions*: configura a animação executada na exibição do alerta.

Por último, para codificar o clique do mouse em cada botão inserido no alerta, no exemplo da **Figura 3**, os botões com a imagem do disquete e da impressora, basta implementar o evento *OnClick* de cada item da propriedade *Buttons*.

Dica: Para conhecer todas as propriedades e eventos do componente, confira o demo existente na pasta *\jvcl\examples\JvDesktopAlert*.

Listagem 1. Exibindo um alerta no desktop

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  with JvDesktopAlert1 do
  begin
    HeaderText := 'ClubeDelphi';
    MessageText :=
      'Exemplo de mensagem popup com JvDesktopAlert';
    Execute;
  end;
end;
```

Listagem 2. Codificando uma Hotkey

```
procedure TForm1.JvApplicationHotKey1HotKey(
  Sender: TObject);
begin
  Application.BringToFront;
  ShowMessage('Combinação Ctrl + Alt + D pressionada');
end;
```

Listagem 3. Extraindo e carregando um arquivo do executável

```
uses ShellAPI;
...
procedure TForm1.Button1Click(Sender: TObject);
var
  Arquivo: string;
begin
  Arquivo := IncludeTrailingPathDelimiter(
    ExtractFilePath(ParamStr(0)));
  { Coloque aqui a extensão do arquivo
    que você escolheu }
  Arquivo := Arquivo + 'arquivo.doc';
  JvDataEmbedded1.DataSaveToFile(Arquivo);
  ShellExecute(Handle, 'open', PAnsiChar(Arquivo),
    nil, nil, SW_SHOWNORMAL);
end;
```

Criando Hotkeys

Uma *hotkey* é uma combinação de teclas que é capturada pela aplicação mesmo quando ela não está com o foco atual do sistema operacional. Criar uma *Hotkey* com a JVCL é extremamente fácil.

Crie um novo projeto, adicione um *JvApplicationHotKey* (*Jv Applications, Forms*), configure a propriedade *Active* para *True* e defina uma combinação de teclas na propriedade *HotKey*, no nosso exemplo será CTRL+ALT+D.

Em seguida, implemente o evento *OnHotKey* com o código da **Listagem 2**.

Execute a aplicação e selecione outro aplicativo no seu computador. Em seguida, pressione a combinação de teclas definida no componente e veja que o projeto será mostrado, juntamente com a mensagem.

Salvando dados no executável

Muitas vezes precisamos que nossa aplicação acesse recursos externos, como um arquivo texto, imagens, sons etc. Podemos simplesmente distribuir esses arquivos junto com o executável do programa, e carregá-los em *runtime* ou adicioná-los diretamente no arquivo EXE.

Para fazer isso, é necessário fazer uso de arquivos de *resource*, além de usar o compilador da Borland específico para esse fim. Simplificando enormemente essa tarefa, a JVCL oferece o *JvDataEmbedded* (*Jv Non-visual*).

Para exemplificar, crie uma nova aplicação e adicione o componente no formulário. Dê um duplo clique sobre ele e selecione o arquivo que você deseja anexar ao programa, podendo ser de qualquer tipo. Em seguida, coloque um botão no formulário e implemente o seu evento *OnClick* com o código da **Listagem 3**.

Nota: É claro, cuidado com o tamanho do arquivo sendo adicionado, pois, além do processo de inclusão/extração demorar, o executável do programa pode aumentar consideravelmente de tamanho.

Nota: Lembre-se de trocar a extensão do arquivo por aquela que você utilizou.

A função *IncludeTrailingPathDelimiter* garante que haverá uma "\" (*contrabarra*) ao final da *string*. Isso é importante para montar um caminho válido de arquivo. Execute a aplicação e clique no botão. O arquivo será extraído do executável da aplicação e aberto com o programa associado a ele.

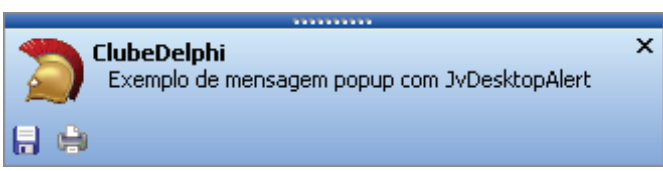


Figura 3. Alerta sendo exibido no desktop

Dica: Você também pode extrair um recurso para a memória a fim de utilizá-lo no próprio aplicativo. Para isso, use o método *DataSaveToStream* ao invés de *DataSaveToFile*.

Trocando Enter por Tab automaticamente

Para fazer com que a tecla ENTER se comporte como TAB, basta adicionar no formulário o *JvEnterAsTab* (*Jv Non-Visual*) e configurar a propriedade *EnterAsTab* para *True* (o padrão na verdade já é *True*).

É importante lembrar que isso apenas funciona no formulário onde o componente foi criado (adicionado) e não em toda a aplicação.

Hints em forma de balão

Você já deve ter notado que nas janelas de senha do Windows XP, quando se pressiona a tecla CAPS LOCK, um *hint* em forma de balão aparece para alertar o usuário sobre o fato. Para conseguir o mesmo efeito, com uma série de customizações, temos o *JvBalloonHint* (*Jv Non-Visual*).

Crie uma nova aplicação e adicione o componente ao formulário. Em seguida, adicione um *Button*, um *Panel*, um *ComboBox* e um *CheckBox*. Codifique o evento *OnClick* do botão como mostrado na **Listagem 4**.

Compile e execute a aplicação. Você deve ver algo parecido com a **Figura 4**.

Entenda o que cada parâmetro utilizado no *ActivateHint* quer dizer:

- *Button1*: o controle onde o balão aparecerá sobre, embaixo, ao lado etc.;
- *Hint em forma...* : a mensagem que vai aparecer no balão;
- *ikWarning*: o tipo do balão. Os valores possíveis são (além dos utilizados no exemplo):
  - *ikCustom*: imagem customizada;
  - *ikApplication*: ícone da aplicação;
  - *ikNone*: não exibe nenhuma imagem.
- *TJvBallonHint*: a mensagem que vai aparecer no cabeçalho do balão;
- *3000*: o tempo, em milisegundos, que o balão permanecerá exibido.

Existem ainda algumas propriedades do componente que merecem ser comentadas:

- *CustomAnimationStyle:* configura um efeito para a exibição do balão;
- *DefaultBalloonPosition:* define a posição onde o balão aparecerá, relativa ao controle especificado no *ActivateHint*;
- *Images:* associa um *ImageList* ao componente, para ser possível usar imagens customizadas nos balões. Nesse caso, troque o terceiro parâmetro do *ActiveHint* pelo índice da imagem desejada.
- *Options:* configura uma série de opções de exibição, como por exemplo, tocar um som quando o balão for exibido;
- *UseBalloonAsApplicationHint:* faz com que todos os *hints* da aplicação sejam exibidos no formato de balão.

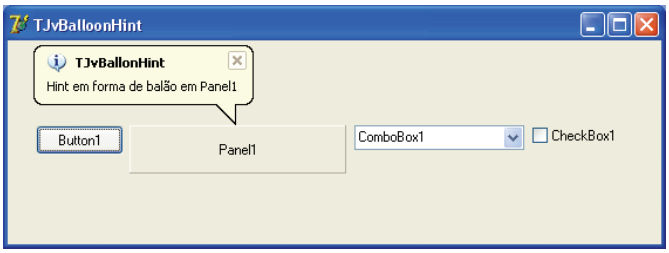


Figura 4. Hints em forma de balão

Listagem 4. Exibindo programaticamente hints no formato de balão

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  with JvBalloonHint1 do
  begin
    ActivateHint(Button1,
      'Hint em forma de balão em Button1', ikWarning ,
      'TJvBallonHint', 3000);
    Sleep(3000);
    ActivateHint(Panel1,
      'Hint em forma de balão em Panel1',
      ikInformation, 'TJvBallonHint', 3000);
    Sleep(3000);
    ActivateHint(ComboBox1,
      'Hint em forma de balão em ComboBox1',
      ikQuestion, 'TJvBallonHint', 3000);
    Sleep(3000);
    ActivateHint(CheckBox1,
      'Hint em forma de balão em CheckBox1', ikError,
      'TJvBallonHint', 3000);
  end;
end;
```

Dica: Confira o demo que acompanha o componente, na pasta *\jvcl\examples\JvBalloonHint*, para conhecer todas as opções de customização disponíveis.

Barra de progresso animada

Existem tarefas, que executamos com certa freqüência em nossos programas, onde não é possível colocar uma barra de progresso para dar um *feedback* visual ao usuário do andamento da operação.

Inserções em banco de dados, execuções de *Stored Procedures* etc., são alguns exemplos. Para essas situações, existe o *JvWaitingGradient* (*Jv Bars, Panels*), que exibe uma barra animada ao usuário.

Crie uma nova aplicação e adicione o componente ao formulário. Configure as propriedade *AlwaysRestart* para *True*, *EndColor* para *clBlue* e *Interval* para 10. Coloque também um botão no formulário, implemente seu evento *OnClick* e digite o seguinte código:

```
JvWaitingGradient1.Active := not JvWaitingGradient1.Active;
```

Execute a aplicação e clique no botão. Veja o resultado na **Figura 5.**

Controles com formatação HTML

A JVCL oferece alguns componentes muito úteis na criação de interfaces gráficas. Como o título desse tópico diz, são controles que aceitam formatação básica em HTML. São eles:

- *JvHTLabel* (*Jv Labels*);
- *JvHTButton* (*Jv Buttons*);

- *JvHTComboBox* (*Jv List, Combos, Trees*);
- *JvHTListBox* (*Jv List, Combos, Trees*).

Adicione os componentes mostrados anteriormente, no formulário, e configure-os como a seguir:

JvHTLabel1.Caption

```
<hr><font size=12><b>JVCL</b> e <b><font color=clRed>
Delphi</font></b> </font><br><br><u><font color=clBlue>
Turbinando suas aplicações</font></u><br><hr>
```

JvHTButton1.Caption

```
<b><font color=clGreen>Delphi</font> rocks!</b>
```

JvHTComboBox1.Items:

```
\<b><font color=clRed>Opção</font> <u>1</u></b>
<b><font color=clRed>Opção</font> <u>2</u></b>
<b><font color=clRed>Opção</font> <u>3</u></b>
<b><font color=clRed>Opção</font> <u>4</u></b>
<b><font color=clRed>Opção</font> <u>5</u></b>
```

JvHTListBox1.Items:

```
<b><font color=clBlue>Item</font> <u>A</u></b>
<b><font color=clBlue>Item</font> <u>B</u></b>
<b><font color=clBlue>Item</font> <u>C</u></b>
<b><font color=clBlue>Item</font> <u>D</u></b>
<b><font color=clBlue>Item</font> <u>E</u></b>
```

Ainda em *designtime* é possível ver o resultado final de alguns controles. Para visualizar todos em ação, execute a aplicação (**Figura 6**).

Confira na **Tabela 1** as *tags* suportadas pelos componentes.

Tag	Descrição	Exemplo
B	Negrito	Texto
I	Itálico	Texto
U	Sublinhado	<u>Texto</u>
S	Tachado	<s>Texto</s>
A	Hyperlink	ClubeDelphi
ALIGN	Alinhamento horizontal	<align=center/right>Texto</align>
FONT	Formatação de fonte	Texto Texto Texto
IND	Indentação	<ind=10>Texto</ind>
BR	Quebra de linha	Linha 1 Linha 2
HR	Linha horizontal	<hr>Texto<hr>

Tabela 1. Tags suportadas pelos componentes da JVCL

Adicionando itens ao menu de sistema

Incluir novos itens no menu de sistema (aquele que aparece quando clicamos de direita sobre a barra de títulos da aplicação ou sobre o seu botão na barra de tarefas) é muito fácil com a JVCL. Basta usar o *JvSystemPopup* (*Jv Non-Visual*), e proceder como um menu qualquer.

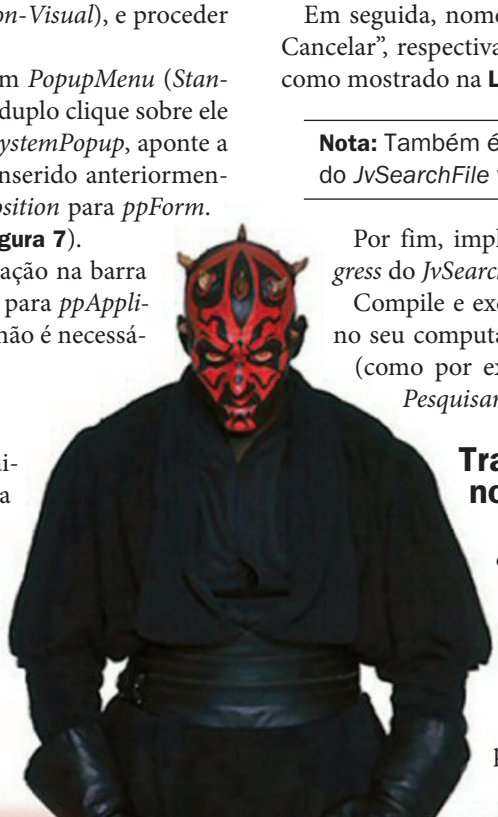
Crie uma nova aplicação e adicione um *PopupMenu* (*Standard*) ao formulário. Em seguida, dê um duplo clique sobre ele e crie alguns itens. Agora, adicione o *JvSystemPopup*, aponte a propriedade *Popup* para o *PopupMenu* inserido anteriormente e configure também a propriedade *Position* para *ppForm*. Execute a aplicação e veja o resultado (**Figura 7**).

Para alterar o menu do botão da aplicação na barra de tarefas, defina a propriedade *Position* para *ppApplication*. Observe que em ambos os casos, não é necessário digitar nenhuma linha de código.

Pesquisando arquivos

Não é raro precisarmos pesquisar arquivos em nossas aplicações e realizar essa tarefa envolve conhecer certas funções não tão óbvias, como *FindFirst* e *FindNext*. A solução da JVCL para esse tipo de problema é o *JvSearchFiles* (*Jv Non-Visual*).

Para demonstrar o seu uso, crie uma nova aplicação e adicione os seguintes componentes ao formulário: um *JvSearchFiles*, um *JvDirectoryEdit* (*Jv Edits*), um *Edit*, um *ListBox*, dois *Button*, dois *Label* e um *StatusBar*. Organize-os visualmente como mostrado na **Figura 8**.



> **Dica:** O *JvDirectoryEdit* é um *Edit* que permite ao usuário selecionar uma pasta do computador, através de um botão no controle. Ele também tem o recurso de auto-completar embutido, e, dessa forma, basta você digitar o início do caminho para que ele exiba uma lista de pastas que comecem com a *string*.

Em seguida, nomeie os botões para "btnPesquisar" e "btnCancelar", respectivamente, e codifique seus eventos *OnClick* como mostrado na **Listagem 5**.

> **Nota:** Também é possível configurar as propriedades do *JvSearchFile* via *Object Inspector*.

Por fim, implemente os eventos *OnFindFile* e *OnProgress* do *JvSearchFiles* com o código da **Listagem 6**.

Compile e execute a aplicação. Selecione alguma pasta no seu computador e informe uma máscara de arquivos (como por exemplo, "*.pas"). Em seguida, clique em *Pesquisar* e veja o resultado (**Figura 9**).

Trabalhando com a área de notificações

Existem muitos componentes para adicionar um ícone à área de notificações do Windows, também conhecida como bandeja ou *System Tray*, mas poucos são tão completos quanto o *JvTrayIcon* (*Jv Non-Visual*).

Crie um novo projeto e adicione o componente ao formulário. Em seguida, confi-

gure a propriedade *Active* para *True* e execute a aplicação. Note que apenas isso já é suficiente para aparecer um ícone na bandeja do sistema. Vamos agora sofisticar um pouco as coisas.

Por padrão, o componente utiliza o ícone da aplicação. Para alterá-lo, selecione uma nova imagem através da propriedade *Icon*. Para não precisar ficar compilando a aplicação para acompanhar as mudanças, altere para *True* o atributo *tvVisibleDesign* da propriedade *Visibility*. Isso mostrará em tempo de design as alterações realizadas.

> **Nota:** Quando executar o aplicativo, lembre-se de desligar esse *flag*, para que não apareçam dois ícones iguais na bandeja do sistema.

Adicione um *PopupMenu* ao formulário e insira três itens de menu. Configure o *Caption* de cada um para "Minimizar", "Restaurar" e "Exibir mensagem", respectivamente. Codifique o evento *OnClick* de cada opção com o código da **Listagem 7**.

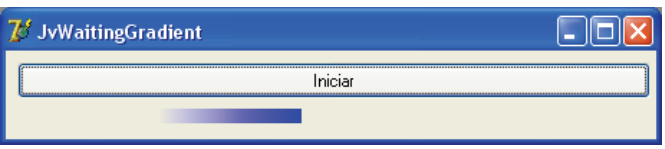


Figura 5. Barra de espera em execução

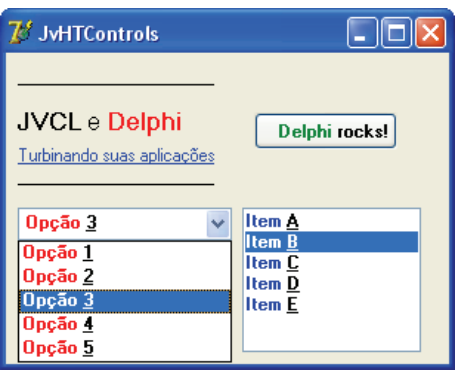


Figura 6. Controles com formatação HTML da JVCL

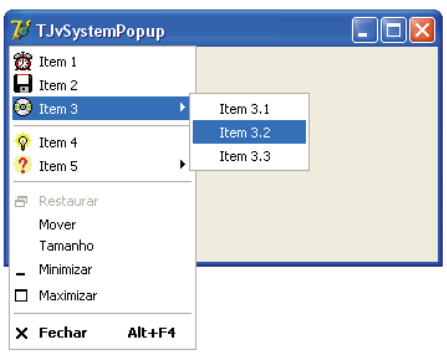


Figura 7. Menu do sistema customizado

Listagem 5. Código dos botões btnPesquisar e btnCancelar

```
procedure TForm1.btnPesquisarClick(Sender: TObject);
begin
  Screen.Cursor := crHourGlass;
  btnPesquisar.Enabled := False;
  btnCancelar.Enabled := True;
  ListBox1.Clear;
  try
    with JvSearchFiles1 do
    begin
      Directories.Clear;
      Files.Clear;
      RootDirectory := JvDirectoryEdit1.Text;
      FileParams.FileMasks.Text := Edit1.Text;
      { Faz pesquisar os arquivos segundo a
       máscara informada }
      FileParams.SearchTypes := [stFileMask];
      { Faz pesquisar também nas subpastas }
      DirOption := doIncludeSubDirs;
      Options := [soSearchFiles, soOwnerData];
      Search;
    end;
  finally
    StatusBar1.SimpleText := Format(
      '%d arquivo(s) encontrados',
      [ListBox1.Items.Count]);
    Screen.Cursor := crDefault;
    btnPesquisar.Enabled := True;
    btnCancelar.Enabled := False;
  end;
end;

procedure TForm1.btnCancelarClick(Sender: TObject);
begin
  { Aborta a pesquisa }
  JvSearchFiles1.Abort;
end;
```

Listagem 6. Eventos OnFindFile e OnProgress

```
uses Math;
...
procedure TForm1.JvSearchFiles1FindFile(
  Sender: TObject; const AName: string);
begin
  with ListBox1 do
  begin
    Items.Add(AName);
    { Exibe a barra de rolagem horizontal se algum
      item do Listbox for maior do que a largura do
      componente }
    ScrollWidth := Max(ScrollWidth,
      (Canvas.TextWidth(AName) + Width -
      ClientWidth));
  end;
  StatusBar1.SimpleText := AName;
  StatusBar1.Update;
end;

procedure TForm1.JvSearchFiles1Progress(
  Sender: TObject);
begin
  { Evita que a aplicação "congele" durante a
   pesquisa }
  Application.ProcessMessages;
end;
```

Listagem 7. Código dos menus de contexto do componente

```
procedure TForm1.MinimizarClick(Sender: TObject);
begin
  { Menu Minimizar }
  JvTrayIcon1.HideApplication;
end;

procedure TForm1.RestaurarClick(Sender: TObject);
begin
  { Menu Restaurar }
  JvTrayIcon1.ShowApplication;
end;

procedure TForm1.ExibirMensagemClick(Sender: TObject);
begin
  { Menu Exibir mensagem }
  JvTrayIcon1.BalloonHint('JvTrayIcon',
    'Mensagem exibida pelo componente. JVCL rules!',
    btWarning);
end;
```

Dica: O *BalloonHint* é muito parecido com *ActivateHint*, do *JvBalloonHint*. Apenas mude *ikError*, *ikQuestion*, etc. por *btError*, *btQuestion*...

Por fim, aponte o *PopupMenu* para a propriedade *Popup* do *JvTrayIcon* e altere para *True* o atributo *tvAnimateToTray* na propriedade *Visibility*. Isso produzirá uma animação quando a aplicação for minimizada/restaurada pelo componente. Compile, execute o projeto e teste as três opções do menu (**Figura 10**).

Para fechar esse tópico, vamos conhecer algumas outras propriedades do componente que podem ser úteis:

• *Animated*: Anima o ícone na área de notificações. Para funcionar, deve-se associar um *ImageList*, contendo as imagens em seqüência para produzir a animação, à propriedade *Icons*.

• *Delay*: Especifica o tempo entre a troca das imagens, para formar a animação;

• *Icons*: Como mencionado anteriormente, aponta para um *ImageList* com as imagens desejadas para o efeito;

• *Hint*: Configura um texto que vai aparecer sobre o ícone na bandeja do sistema quando o mouse estiver sobre ele.

Veremos mais propriedades do componente no próximo tópico.

Adicionando botões à barra de títulos de formulários

Existem certas funcionalidades que dão um toque profissional a uma aplicação. Uma delas, com certeza, é o que o *JvCaptionButton* (*Jv Buttons*) faz: adicionar botões customizados à barra de títulos do formulário.

Crie uma nova aplicação e adicione ao formulário o componente. Observe que, ao fazer isso, um botão aparecerá na barra de títulos do formulário, ainda em *designtime*, à esquerda do *Minimizar*.

Defina as propriedades *Standard* para *tsbMinimizeToTray* e *Hint* para "Minimizar para a área de notificações". Adicione também um *JvTrayIcon*, do tópico anterior. Configure suas propriedades dessa forma: *Active*: *True*, *Visibility.tvRestoreDblClick*: *True*; *Visibility.tvAnimateToTray*: *True*.

Agora, implemente o evento *OnClick* do *JvCaptionButton*, com o seguinte código:

```
JvTrayIcon1.HideApplication;
```

Pronto. Execute a aplicação e clique no novo botão na barra de títulos do formulário (**Figura 11**). Para restaurar a aplicação, dê um duplo clique sobre o seu ícone na área de notificações e veja o objetivo da propriedade *Visibility.tvRestoreDblClick*.

Agora, insira outro *JvCaptionButton* ao formulário. Configure as seguintes propriedades: *Caption*: "ClubeDelphi"; *Alignment*: *taCenter*; *ButtonWidth*: 100; *Position*: 1; *ImageIndex*: 0 e *Font.Color*: *clWindow*. Adicione um *ImageList* ao formulário e inclua nele uma imagem qualquer.

Em seguida, aponte a propriedade *Images* do componente para esse *ImageList* e configure o atributo *ImageIndex* para um valor válido (0 ou maior, dependendo da quantidade de imagens colocadas no *ImageList*). Compile o projeto e veja o resultado na **Figura 12.**

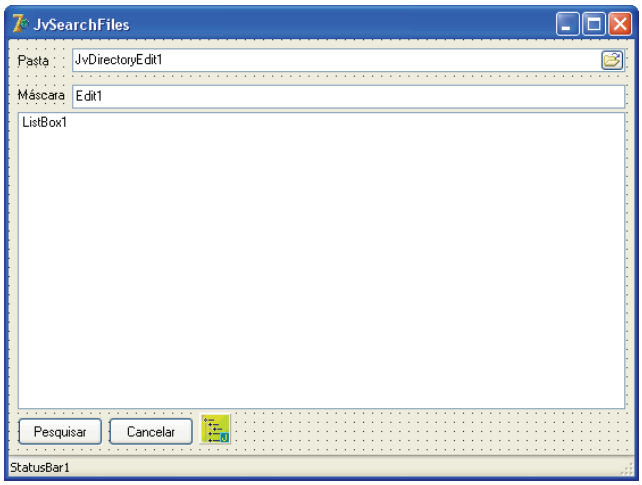


Figura 8. Tela de pesquisa de arquivos

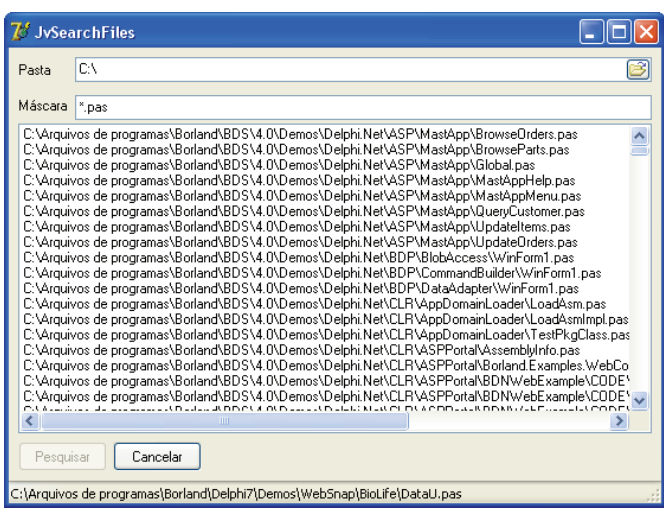


Figura 9. Pesquisando arquivos com o JvSearchFiles

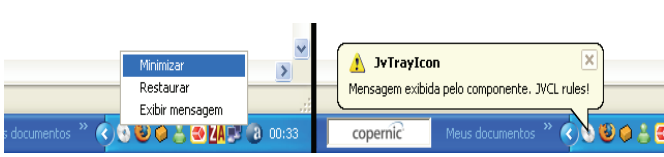


Figura 10. Menu pop-up e mensagem sendo exibidos na bandeja do sistema

Treeview com CheckBoxes

Uma coisa simples e rotineira, que não existe nativamente no *TreeView* que acompanha o Delphi, é adicionar *CheckBoxes* aos itens da árvore. Para conseguir esse efeito, sem nenhum esforço, podemos utilizar o *JvCheckTreeView* (*Jv Lists, Combos, Trees*).

Crie um novo projeto, adicione o componente ao formulário e modifique a propriedade *Checkboxes* para *True*. Em seguida, dê um duplo clique sobre o componente e insira alguns itens e

sub-itens. Por fim, adicione um botão, altere seu *Caption* para "Mostrar selecionados" e implemente seu evento *OnClick* para o código da **Listagem 8**.

Listagem 8. Verificando os itens marcados no JvCheckTreeView

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
  S: string;
  Node: TTreeNode;
begin
  S := 'Itens selecionados:' + #13 + #13;
  for I := 0 to JvCheckTreeView1.Items.Count - 1 do
  begin
    Node := JvCheckTreeView1.Items[I];
    if JvCheckTreeView1.Checked[Node] then
      S := S + Node.Text + #13;
  end;
  ShowMessage(S);
end;
```

Execute a aplicação, selecione alguns itens e clique no botão (**Figura 13**).

As mesmas coisas que você está acostumado a fazer com o *Treeview* nativo do Delphi podem ser feitas com esse componente, como associar um *ImageList* para exibir imagens para cada item por exemplo.

Para obter mais detalhes, consulte o demo que acompanha a JVCL, disponível em *\jvcl\examples\JvCheckTreeView*.

> **Dica:** Se você precisar de um *Treeview* que suporte muitos itens e tenha recursos avançados, consulte na edição 71 da revista ClubeDelphi o artigo sobre o *VirtualTreeView*.

Controles "linkados"

Quando projetamos interfaces gráficas (GUI's), frequentemente nos deparamos com a seguinte situação: a ativação de um controle ou de um grupo de controles depende de um *Checkbox*/*Radiobutton* selecionado. A JVCL oferece uma maneira incrivelmente fácil de conseguir isso, sem uma única linha de código.

Inicie uma nova aplicação e adicione no formulário os seguintes componentes: dois *JvCheckBox* (*Jv Buttons*), dois *Edit*, dois *JvRadioButton* (*Jv Buttons*), quatro *Label*. Organize-os no formulário como mostrado na **Figura 14**.

> **Nota:** Neste exemplo não vamos nos preocupar em dar nomes nem modificar os *Captions* dos componentes.

Agora vamos configurar alguns controles. Selecione o *JvCheckbox1* e dê um duplo clique sobre sua propriedade *LinkedControls*. No editor, insira um novo item e aponte sua propriedade *Control* para *JvCheckBox2* (**Figura 15**).

Agora, selecione o *JvCheckbox2* e repita o processo anterior, com a diferença de adicionar dois itens no editor, apontando suas propriedades *Control* para *Edit1* e *Edit2*, respectivamente.

Refaça esse procedimento para os *JvRadioButtons*, referenciando em *Control* o *Label3* para o primeiro e *Label4* para o

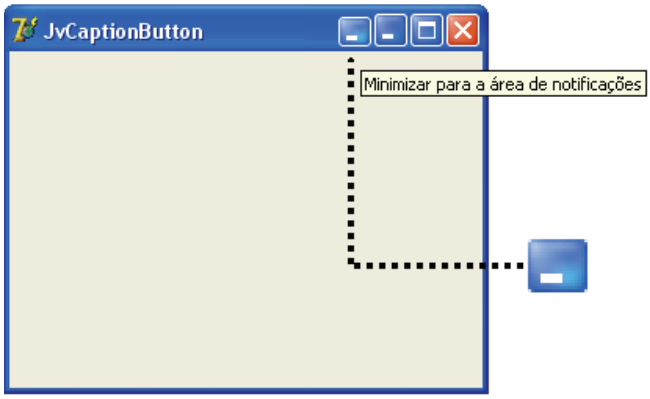


Figura 11. Minimizar extra na barra de títulos do formulário

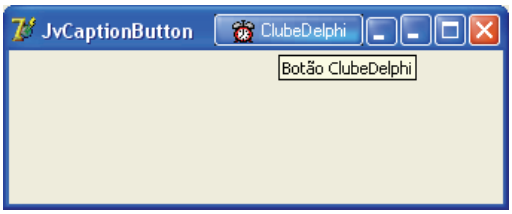


Figura 12. Botão customizado na barra de título do formulário

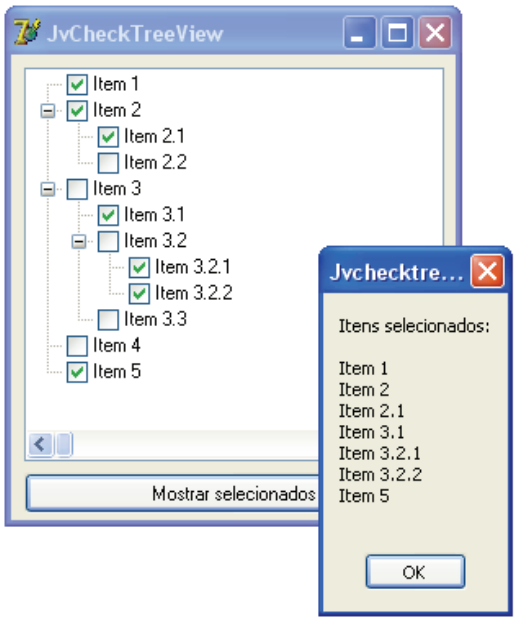


Figura 13. TreeView com Checkboxes

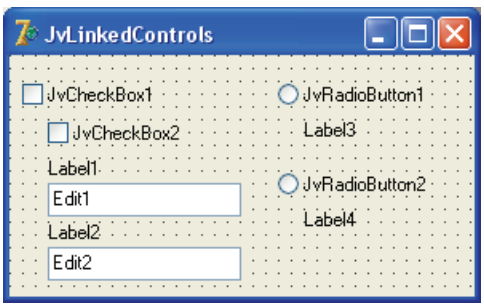


Figura 14. Controles organizados no formulário

segundo. Por fim, defina a propriedade *Checked* do *JvRadioButton1* para *True*.

Repare que a essa altura, o estado dos componentes do formulário deve estar diferente do original. Execute a aplicação e veja o resultado (**Figura 16**).

Note que ligamos o estado de controles às propriedades de outros, realizando o proposto no início desse tópico, sem digitar uma única linha sequer de código.

Validando formulários

Validar os dados inseridos pelo usuário em um formulário é tarefa obrigatória em qualquer sistema. Para isso, existem diversas maneiras, seja codificando individualmente algum evento de cada controle ou utilizando rotinas para automatizar o processo.

A JVCL oferece uma maneira bastante otimizada e elegante de validar controles. Para conhecer como ela funciona, crie uma nova aplicação e construa o formulário de acordo com a **Figura 17**.

Apague o conteúdo da propriedade *Text* de todos os *Edits* e configure a propriedade *PasswordChar* de *Edit3* e *Edit4* para "*". Em seguida, inclua no formulário os componentes *JvValidators*, *JvErrorIndicator* e *JvValidationSummary* (*Jv Validators*).

Dê um duplo clique no *JvValidators1* para abrir o editor de validações. Clique na seta ao lado do primeiro botão e selecione *Required Field Validator*. Configure as propriedades *ControlToValidate* para *Edit1*, *ErrorMessage* para "Informe um nome" e *PropertyToValidate* para *Text*.

> **Nota:** Estamos definindo que o primeiro *Edit* não poderá ficar em branco, ele é um campo requerido.

Clique novamente na seta ao lado do primeiro botão do editor, selecione *Controls Compare Validator* e defina suas propriedades: *CompareToControl*: *Edit3*; *CompareToProperty*: "Text" e *ControlToValidate*: *Edit4*. Finalize alterando, *ErrorMessage* para "A senha de confirmação deve ser igual a anterior", *Operator*: *vcoEqual* e *PropertyToValidate*: *Text*.

> **Nota:** Estamos dizendo que os controles *Edit3* e *Edit4* devem ter o mesmo valor em suas propriedades *Text*.

Por fim, volte ao editor e adicione um *Custom Validator*. Configure a propriedade *ControlToValidate* para *Edit2*, *ErrorMessage* para "Informe um e-mail válido" e *PropertyToValidate* para *Text*. Em seguida, implemente o evento *OnValidate* com o código da **Listagem 9**.

> **Nota:** Criamos um *validator* personalizado, que fará uma verificação simples, através do evento *OnValidate*, se o e-mail digitado é válido.

Ligue ao *JvValidator* os outros dois: *JvErrorIndicator* e *JvValidationSummary*. Agora, implemente o código do evento *OnClick* do botão e digite o código da **Listagem 10**.

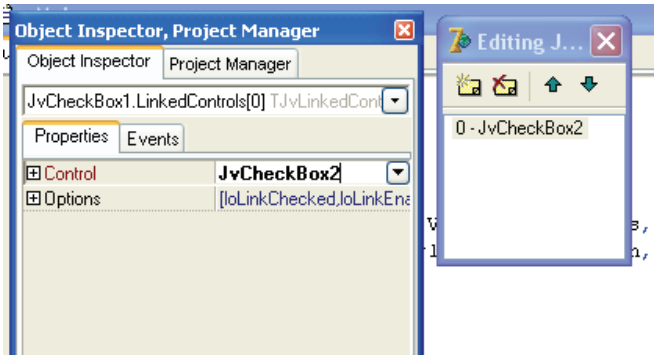


Figura 15. Configurando um item da propriedade LinkedControls

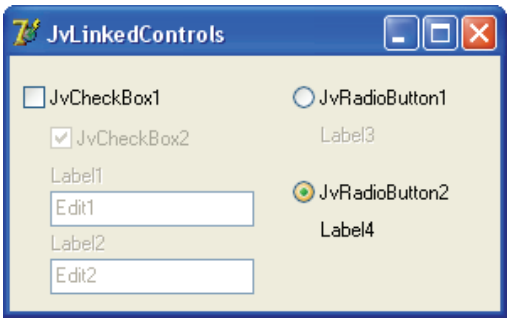


Figura 16. Controles ativados em função de outros

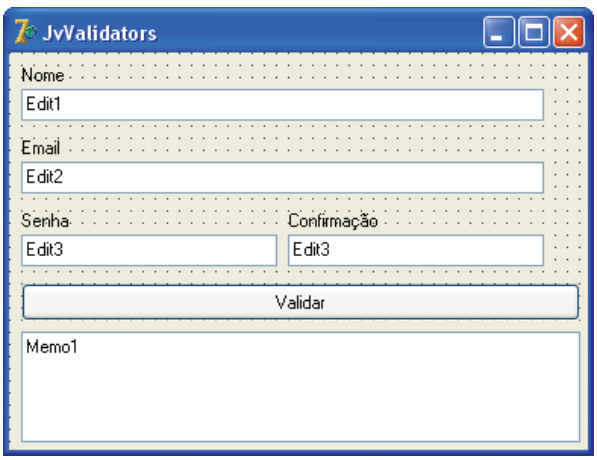


Figura 17. Formulário simples de cadastro

Listagem 9. Código do evento OnValidate

```
procedure TForm1.JvCustomValidator1Validate(
  Sender: TObject; ValueToValidate: Variant;
  var Valid: Boolean);
begin
  Valid := (Pos('.', ValueToValidate) > 0) and
    (Pos('@', ValueToValidate) > 0);
end;
```

Listagem 10. Executando os validadores criados

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  JvErrorIndicator1.BeginUpdate;
  try
    JvErrorIndicator1.ClearErrors;
    if not JvValidators1.Validate then
      Memo1.Lines.Text :=
        JvValidationSummary1.Summaries.Text;
  finally
    JvErrorIndicator1.EndUpdate;
  end;
end;
```

O *Validate*, da classe *TJvValidators* executa os validadores que criamos anteriormente. Para cada validação que falhar, o *JvErrorIndicator* exibirá um pequeno alerta ao lado do controle associado. Como temos vários campos para serem validados, precisamos chamar os métodos *BeginUpdate* e *EndUpdate* da classe *TJvErrorIndicator* para evitar um efeito de dessincronia entre os alertas exibidos.

CleanErrors esconde as imagens indicativas, pois se algum campo se tornar válido após uma validação (o usuário preencheu o primeiro *Edit*, por exemplo), o seu alerta deve desaparecer. Por fim, o *JvValidationSummary* guarda todas as mensagens de erro que ocorreram, e as exibimos no *Memo* existente no formulário.

Como uma imagem vale mais que mil palavras, dê uma olhada na **Figura 18**.

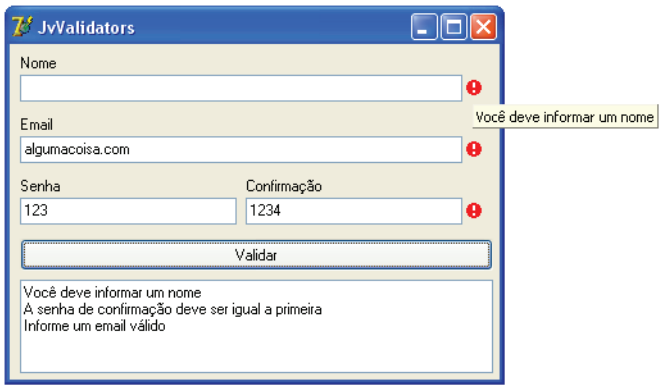


Figura 18. Validadores em ação

Nota: Existe um *validador* específico para expressões regulares, que seria mais adequado para validarmos o e-mail digitado. Porém, em nossos testes, o componente aparentemente apresentou alguns problemas. Acredito que em futuras versões da biblioteca os problemas tenham sido resolvidos.

Dica: Você pode alterar a imagem utilizada para sinalizar um erro. Para isso, ligue um *ImageList* ao *JvErrorIndicator* e selecione o índice desejado na propriedade *ImageIndex*.

Persistindo propriedades de formulários

É comum em um sistema que as janelas utilizadas pelo usuário mantenham seu estado quando são fechadas, e o restaure novamente quando for exibida uma segunda vez. Por estado, entendemos tamanho, posição etc. A JVCL disponibiliza uma solução bastante simples para isso, que veremos agora.

Inicie uma nova aplicação, e adicione ao formulário os componentes *JvFormStorage* e *JvAppRegistryStorage* (*Jv Persistence*). Ligue-os através da propriedade *AppStorage*, presente no *JvFormStorage*. Em seguida, configure a propriedade *Root*, de *JvAppRegistryStorage*, para "ClubeDelphi".

Pronto. Compile e execute a aplicação, mova o formulário de lugar, redimensione-o, feche-o e rode o programa novamente. Repare que todas as modificações que você fizer serão persistidas e restauradas automaticamente.

Como utilizamos o *JvAppRegistryStorage*, os dados do formulário foram salvos no registro do Windows. Se você preferir, pode utilizar um arquivo INI (*JvAppIniFileStorage*), o banco de dados (*JvAppDBStorage*) ou ainda um arquivo XML (*JvAppXMLFileStorage*), com poucas modificações entre eles.

Veja na **Figura 19** como os dados são salvos.

Criando assistentes

Existem muitas situações em que um assistente é a melhor maneira de tornar simples uma tarefa complexa para o usuário.

ClubeDelphi PLUS

Acesse agora mesmo o Portal do Assinante ClubeDelphi e assista a uma vídeo-aula de Luciano Pimenta que mostra como trabalhar com todos os controles de validação do JEDI.

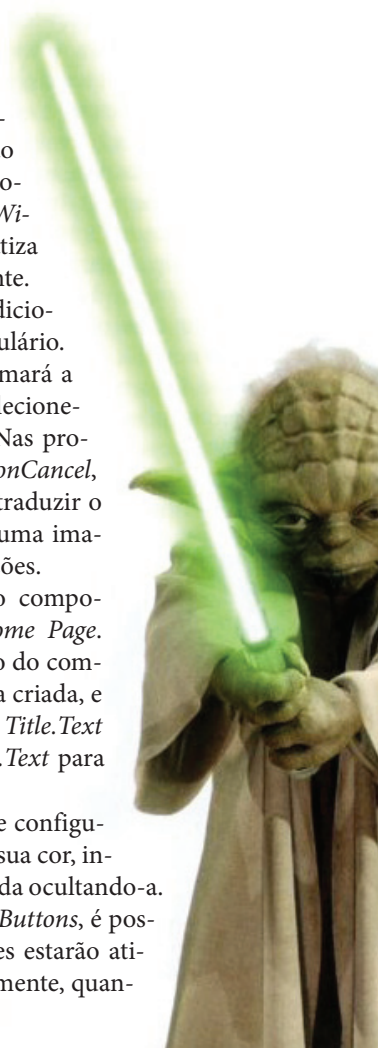
www.devmedia.com.br/articles/viewcomp.asp?comp=3752

Criar um "na mão" envolve certo trabalho manipulando botões, abas etc., fora o esforço de codificar a transição entre as páginas. Para esse problema, a JVCL oferece o *JvWizard* (*Jv Wizard*), que automatiza toda a criação de um assistente.

Crie um novo projeto e adicione o componente ao formulário. Note que o formulário já tomará a aparência de um *wizard*. Selecione-o e vá ao *Object Inspector*. Nas propriedades *ButtonBack*, *ButtonCancel*, *ButtonFinish* etc., podemos traduzir o texto dos botões, adicionar uma imagem, entre outras configurações.

Clique de direita sobre o componente e escolha *New Welcome Page*. Em seguida, clique no centro do componente, sobre a nova página criada, e modifique as propriedades *Title.Text* para "Bem vindo" e *Subtitle.Text* para "JvWizard rocks!".

Em *WaterMark*, você pode configurar a barra lateral, alterando sua cor, inserindo uma imagem, ou ainda ocultando-a. Em *EnabledButtons* e *VisibleButtons*, é possível determinar quais botões estarão ativados ou visíveis, respectivamente, quan-



do a página se tornar a atual no assistente. Você pode adicionar controles livremente sobre a página criada, a organizá-los da forma que achar melhor.

Clique novamente de direita sobre o *JvWizard* e escolha *New Interior Page*, para criar uma página simples dentro do *wizard*. Configure-a exatamente como a anterior, exceto pela marca d'água, que não existe nesse tipo de página.

Defina o seu título como "JVCL JvWizard" e seu subtítulo como "Criando assistentes num piscar de olhos!". Novamente, adicione os controles que desejar sobre ela.

Dica: Para navegar entre as páginas criadas, use os botões *Avançar* e *Voltar*, do próprio componente, ainda em *designtime*.

Execute a aplicação e veja o resultado (**Figura 20**). Observe que criamos um assistente sem nenhuma linha de código digitada.

Impedindo múltiplas instâncias da aplicação

Existem muitas formas de se impedir que uma aplicação seja executada mais de uma vez, e uma delas é com o *JvAppInstances* (*Jv System*). Em uma nova aplicação, adicione o componente ao formulário e configure na propriedade *MaxInstances* quantas cópias do aplicativo poderão ser executadas ao mesmo tempo, no nosso exemplo, defina o valor 1.

Assegure-se que a propriedade *Active* do componente está definida com *True* e compile a aplicação. Agora, abra a pasta onde o arquivo executável foi gerado, e tente executá-lo mais de uma vez. Observe que, após a primeira cópia carregar, as demais apenas a ativarão.

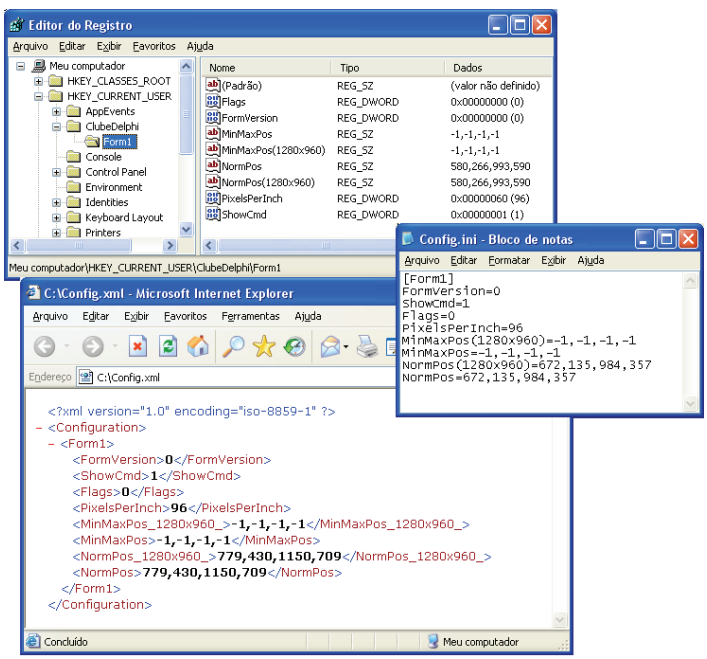


Figura 19. Configurações do registro do Windows, em um arquivo INI e XML

Janelas de configuração

É comum todo software possuir uma janela com as suas configurações, alguns chamam também de "opções" ou "parâmetros". A maneira de projetar essa janela foi por muito tempo baseada em abas, que agrupavam as opções segundo critérios lógicos.

Recentemente, esse modelo sofreu mudanças, e muitos aplicativos substituíram as guias por elementos em um *Treeview*, de maneira a hierarquizar as configurações. Implementar esse design com a JVCL é extremamente fácil, como veremos a seguir.

Crie uma nova aplicação e adicione um *JvPageListTreeView* (*Jv Lists, Combos, Trees*). Ele é um *Treeview* muito semelhante ao nativo do Windows, portanto você que está acostumado a manipular o componente, não deve ter dificuldades.

Configure sua propriedade *Align* para *alLeft*, e em seguida dê um duplo clique sobre o controle e monte a estrutura hierárquica mostrada na **Figura 21**.

Modifique a propriedade *ShowButtons* para *True*, a fim de exibir os botões de expansão/contração dos nós da árvore. Adicione ao formulário um *JvPageList* (*Jv Lists, Combos, Trees*) e configure sua propriedade *Align* para *alClient*.

Clique duas vezes sobre ele e no editor, adicione doze itens, que representam as páginas do componente, responsáveis nesse exemplo por agrupar cada opção representada pelos elementos do *Treeview*.

Em seguida, inclua um *Label* à primeira página criada, e modifique o seu *Caption* para o primeiro item do *TreeView*, nesse caso, "Geral". Repita esse processo para cada página do *JvPageList*.

Dica: Para navegar rapidamente entre as páginas, abra o editor e selecione a página desejada.

Por fim, precisamos vincular ambos os controles. Para isso, primeiramente aponte a propriedade *PageList* do *JvPageListTreeView* para o *JvPageList1*. Depois, clique de direita sobre o *Treeview* e selecione *Links Editor*. Na janela que surgir (**Figura 22**), serão exibidos todos os elementos da árvore e todas as páginas do container.

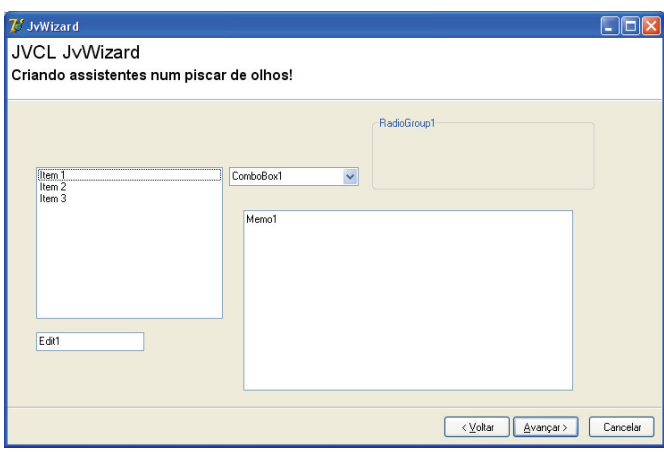


Figura 20. Assistente em execução

Selecione o primeiro item (*Geral*) no painel à esquerda e marque a página correspondente no painel à direta. Em seguida, clique no botão *Create link* para relacionar o elemento com a página. Refaça esse processo para todos os itens do *Treeview*.

Pronto. Novamente sem digitar uma linha sequer de código, execute a aplicação e veja o resultado (**Figura 23**).

Barra de navegação estilo Outlook 2003

Quem usa o Outlook 2003 conhece a barra de navegação, também conhecida como *NavBar*, localizada à esquerda da janela principal do aplicativo, de onde o usuário pode verificar seus e-mails, compromissos ou fazer uma anotação. Para projetar interfaces gráficas como essa, a JVCL oferece os excelentes componentes da paleta *Jv NavPane*. Vejamos alguns deles, na prática.

Inicie um novo projeto e adicione um *JvNavigationPane* ao formulário. Altere sua propriedade *Align* para *alClient* e dê um duplo clique sobre ele para abrir o editor de páginas. Adicione então três páginas, mudando as propriedades *Caption* e *ImageIndex* de cada uma delas para "Email", "0"; "Calendário", "1"; e "Contatos", "2", respectivamente.

Inclua no formulário dois *ImageList*. No primeiro, adicione três imagens com resolução de 16x16 *pixels*. No segundo, se possível, as mesmas imagens, mas em 24x24 *pixels*. Ligue o primeiro *ImageList* à propriedade *SmallImages* do *JvNavigationPane*, e o segundo, à propriedade *LargeImages*.

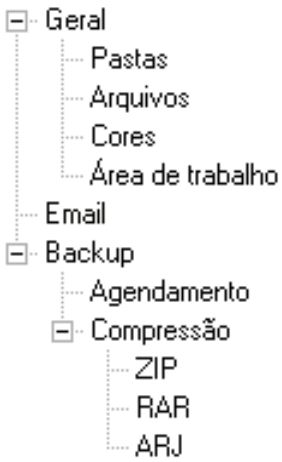


Figura 21. Estrutura de dados do controle JvPageListTreeView

Nota: As imagens de 16px aparecerão na parte inferior do controle, quando os botões de cada página estiverem ocultos. Já as de 24px estarão nos botões principais, e na barra superior do componente (vista a seguir).

Caso você não possua imagens iguais em resoluções diferentes, baixe os exemplos deste artigo e faça uso das mesmas imagens empregadas no exemplo.

Impressão Rápida em Matricias...

Imprimir | Fechar

RDprint 4.0

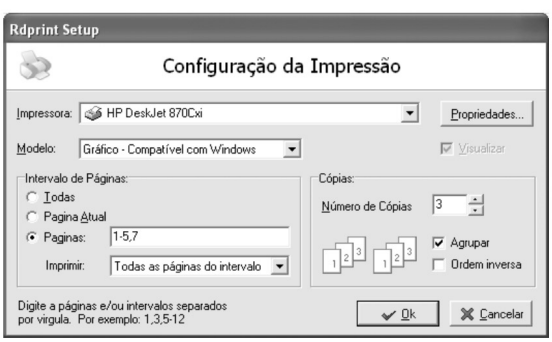
Nova Versão !

O mais completo componente para impressão em MATRICIAIS !
LIDERANÇA absoluta na sua categoria !
Ideal para Notas Fiscais, Duplicatas, Boletos Bancários, etiquetas e relatórios em geral.

- Opção para impressão colorida
- Ajustes de margens para impressão gráfica
- Opção para ocultar a barra de progresso
- Variáveis PAGINAS, DATA, HORA e TÍTULO

Novo form de SETUP com :
- Mapeamento das impressoras e Modelos
- Seleção de páginas igual ao word (1-5,7,8)
- Opção para Inverter e Agrupar cópias na impressão

Novo form de PREVIEW com:
- Função para Procura de TEXTO no relatório
- ROLAGEM com salto automático de página
- ARRASTO da imagem do preview
- StatusBar com informações da impressão
- Novos ícones personalizados



* Disponível para Delphi 5, 6, 7, 2005 e 2006 (VCL)
* Compatível com todas as versões do Windows
* Imprime em portas LPT / COM e USB (modo gráfico)

Deltress Informática

Fone/Fax (14) **3454-7880**
www.deltress.com.br

Página: 2 de 19 | 87% | Impressoa: HP DeskJet 870Cxi | Grafico | * O RDprint 4.0 não imprime gráficos !

Altere agora a propriedade *AutoHeader* do painel de navegação para *True*, a fim de exibir a barra de título superior. A essa altura o seu formulário deve estar parecendo com o da **Figura 24**.

A área em branco que cada página possui é um container de controles, isso é, você pode adicionar quaisquer componentes que desejar. Então, adicione à primeira página um *TreeView*, defina sua propriedade *Align* para *alClient* e clique duas vezes sobre ele e adicione os seguintes itens: "Caixa de entrada", "Caixa de saída", "Itens enviados", "Itens excluídos" e "Rascunhos".

Selecione a página *Calendário* e adicione um *MonthCalendar* (*Win32*). Novamente, ajuste o seu alinhamento para que ocupe toda a área útil disponível. Por fim, selecione a última página, *Contatos* e adicione um *ListBox*. Edite sua propriedade *Items* e adicione alguns nomes de pessoas. Não se esqueça de configurar seu alinhamento para *alClient*.

Para dar o toque final, adicione um *JvNavPaneStyleManager* (*Jv NavPane*), e ligue-o à propriedade *StyleManager* do *JvNavigationPane*. Na propriedade *Theme* (do *JvNavPaneStyleManager*), escolha o tema que mais lhe agradar, portanto basta selecionar um e ver o resultado ainda em *designtime*.

Com tudo pronto, execute a aplicação e veja o controle em ação (**Figura 25**). Repare que, ao arrastar para baixo o divisor que está logo acima dos botões, de modo a escondê-los, outros pequenos botões aparecerão na barra inferior, para ser ainda ser possível navegar entre as páginas.

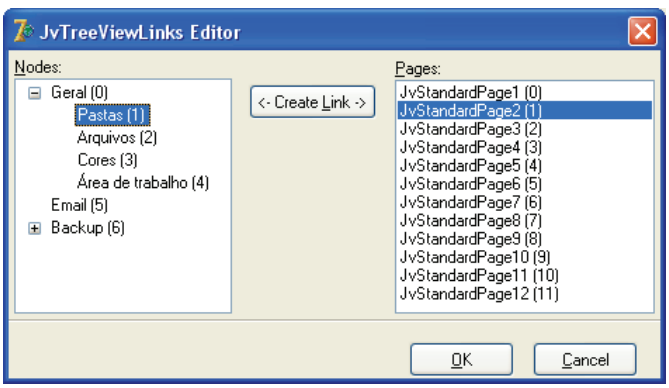


Figura 22. Relacionando os itens do Treeview com as páginas do PageList

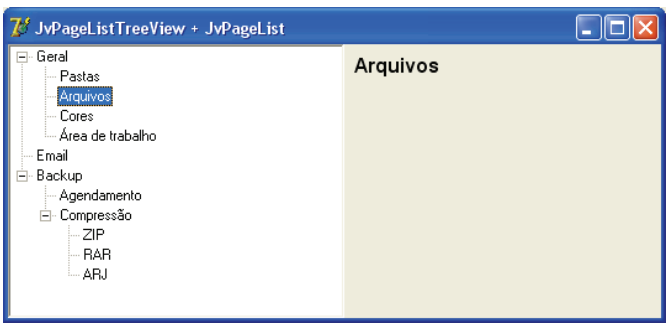


Figura 23. Aplicação TreeView + PageList em execução

Conclusão

Vimos neste artigo um pouco da JVCL, que possui mais de 600 componentes, onde selecionei alguns que julguei interessantes e úteis para o dia a dia do desenvolvedor. Mas, nem de longe, vimos a fundo cada um deles.

Seria necessário um artigo dedicado a cada controle para mostrarmos todas as funcionalidades disponíveis. Apenas fiz as honras e mostrei o caminho das pedras para o leitor. Dessa forma, sugiro estudá-los com afinco, verificar os demos existentes na pasta *jvcl\examples* e garimpar a documentação da biblioteca, que, embora ainda não seja completa, possui dicas valiosas sobre a utilização da JVCL.

Com um pouco de trabalho, podemos poupar muito esforço de codificação. Sucesso em seus projetos com o Delphi, um forte abraço e até a próxima! ■

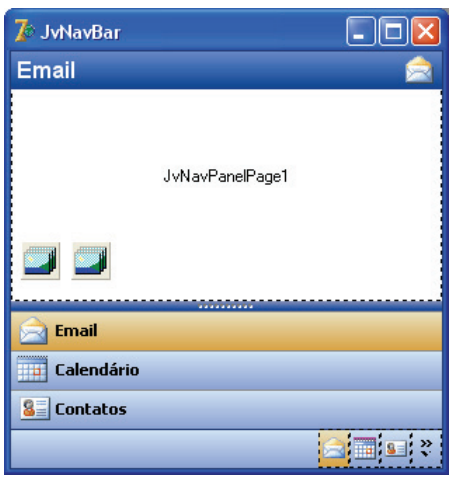


Figura 24. Painel de navegação em designtime

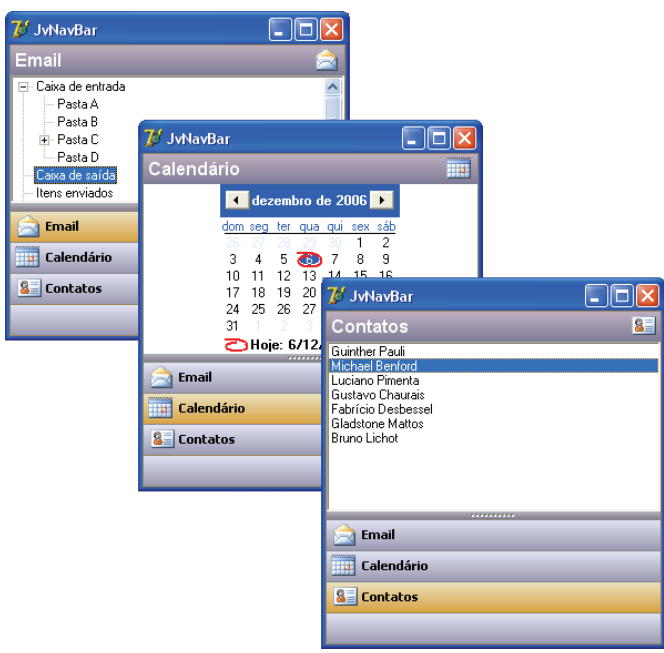


Figura 25. Aplicação com painéis de navegação

Criando Stored Procedures para manipular strings

IVAN PRENOSIL

Funções de *strings* (por exemplo: *Trim*, *Trunc*, *Len* e *SubStr*) podem ser facilmente implementadas por UDFs, contudo, em algumas situações, poderíamos não querer ou poder usar UDFs. Algumas das razões para evitar UDFs são:

- Não são suportadas no Netware;
- São dependentes de plataforma;
- A ausência de UDFs poderia impedir a realização de restauração do banco de dados;
- Não carregam nenhuma informação sobre o conjunto de caracteres dos seus parâmetros;
- Não conseguem retornar *null*;
- UDFs incorretamente escritas podem provocar a queda do servidor.

> **Nota:** Algumas funções aqui apresentadas podem ser nativas em versões mais recentes do banco de dados Firebird, como a 2.0.

TrimRight em SP

O Firebird/InterBase não possui funções nativas para remover espaços à direita. Tal função poderia ser útil, por exemplo, para converter strings CHAR em VARCHAR, importando dados de arquivos externos. A *RTrim* faz parte, por exemplo, da biblioteca de UDFs padrão *ib_udf.dll*, mas como comentamos anteriormente, às vezes, poderíamos querer evitar usar UDFs.

A menos que precisemos usá-la em *strings* muito longas (por exemplo, CHAR(30000)), é possível implementar essa função como um *Stored Procedure*. Quando fazemos o CAST de uma *string* para outra com cumprimento menor, o comando terá sucesso caso os caracteres retirados sejam espaços, caso contrário falhará (levantará uma exceção) quando tentarmos remover caracteres não brancos.

Portanto, podemos resolver o problema fazendo o CAST de uma *string* para outra de menor cumprimento (e assim sucessivamente) interceptando as sucessivas exceções. Poderia ser chamada tanto como um *Stored Procedure* (por exemplo: EXECUTE PROCEDURE TrimRight 'abc') quanto como uma procedure de seleção (como por exemplo: SELECT ... FROM TrimRight ('abc')).

Na **Listagem 1** temos o exemplo da criação de uma SP para utilização do *TrimRight*.

Os exemplos de utilização da SP podem ser vistos na **Listagem 2**.

No código a seguir, temos um exemplo de chamada da SP em outra SP, que executa a *TrimRight* passando uma variável como parâmetro, preenchendo outra variável com o resultado:

```
EXECUTE PROCEDURE TrimRight str_in
RETURNING_VALUES str_out;
```

Não é possível usar um laço WHILE DO para simplificar o código, porque o CAST não pode conter a variável em lugar do comprimento da *string* (isso é: CAST(str AS CHAR(:len))).

Existe um bug no IB 5.1/5.6 (corrigido no IB6), que caso removermos os parênteses internos BEGIN/END, então SELECT FROM TrimRight(null) e SELECT FROM TrimRight(''); retornarão duas linhas em lugar de uma.

Truncamento de string em SP (Trunc)

Diferentemente da remoção, o truncamento é uma função que encurtará a *string* independentemente do seu conteúdo, isso é, removerá inclusive caracteres não brancos. Não existe nenhuma função desse tipo no IB/FB (exceto em UDF externa).

Ao fazermos o CAST de *strings* de maior comprimento para outras de menor, o IB/FB levantará a exceção: "...string truncation". Quando atribuímos uma *string* de maior comprimento diretamente em uma variável de menor comprimento, o IB/FB também levantará a exceção, mas o valor truncado será atribuído de qualquer maneira.

Tudo o que temos a fazer, é interceptar a exceção através da cláusula WHEN ANY DO. Na **Listagem 3**, uma SP de exemplo que trunca a *string* para 5 caracteres.

Para testar, use o seguinte código:

```
EXECUTE PROCEDURE Trunc10To5 '1234567890'
Resultado: '12345'.
```

Note que não devemos utilizar o CAST, logo esses dois procedimentos semelhantes mostrados na **Listagem 4** não funcionarão.

Observe também, que o fato do valor a ser atribuído, mesmo quando a exceção é levantada, possivelmente seja um bug, contudo, essa é a maneira como o IB4, IB5 e IB6 trabalham. Se você utiliza versões mais recentes do IB/FB, não existe a possibilidade, pois a exceção é levantada de qualquer maneira.

A função Length em SP

Como a função *Length* não modifica a *string* de entrada (tal como acontece com as funções de remoção e truncamento), pois retorna o tamanho da *string*, podemos fazer a implementação utilizando diretamente laços do tipo WHILE e LIKE, como podemos ver na **Listagem 5**.

Listagem 1. Criando uma SP para o TrimRight

```
CREATE PROCEDURE TrimRight (str VARCHAR(10))
  RETURNS (ret VARCHAR(10)) AS
BEGIN
  ret = str;
  IF (str IS NULL) THEN
  BEGIN
    SUSPEND;
    EXIT;
  END
  IF (str = '') THEN
  BEGIN
    ret = '';
    SUSPEND;
    EXIT;
  END
  BEGIN
    ret = CAST (str AS char(9));
    ret = CAST (str AS char(8));
    ret = CAST (str AS char(7));
    ret = CAST (str AS char(6));
    ret = CAST (str AS char(5));
    ret = CAST (str AS char(4));
    ret = CAST (str AS char(3));
    ret = CAST (str AS char(2));
    ret = CAST (str AS char(1));
    SUSPEND;
    WHEN ANY DO SUSPEND;
  END
END
```

Listagem 2. Utilização da SP TrimRight

```
SELECT '>' || ret || '<' FROM TrimRight (null)
Resultado: Null

SELECT '>' || ret || '<' FROM TrimRight (' 1234      ')
Resultado:  > 1234<

EXECUTE PROCEDURE TrimRight '1234      '
Resultado: 1234
```

Listagem 3. Stored Procedure que trunca a string

```
CREATE PROCEDURE Trunc10To5 (a VARCHAR(10))
  RETURNS (ret VARCHAR(5)) AS
BEGIN
  ret = '';
  ret = a;
  WHEN ANY DO EXIT;
END
```

Listagem 4. Exemplo que não funciona

```
CREATE PROCEDURE test1 (a VARCHAR(10))
  RETURNS (ret VARCHAR(5)) AS
BEGIN
  ret = null;
  ret = a;
  WHEN ANY DO EXIT;
END

CREATE PROCEDURE test2 (a VARCHAR(10))
  RETURNS (ret VARCHAR(5)) AS
BEGIN
  ret = CAST(a AS VARCHAR(5));
  WHEN ANY DO EXIT;
END
```

Listagem 5. Criando a Stored Procedure para verificar o comprimento de uma string

```
CREATE PROCEDURE Len (str VARCHAR(100))
  RETURNS (len INTEGER) AS
DECLARE VARIABLE pat VARCHAR(100);
BEGIN
  len = null;
  IF (str IS NULL) THEN
    EXIT;
  pat = '';
  len = 0;
  WHILE (NOT str LIKE pat) DO
  BEGIN
    pat = pat || '_';
    len = len + 1;
  END
END
```

Listagem 6. Exemplos utilizando a SP Len

```
EXECUTE PROCEDURE Len null
Resultado: Null

EXECUTE PROCEDURE Len ''
Resultado: 0


EXECUTE PROCEDURE Len 'abc'
Resultado: 3

EXECUTE PROCEDURE Len 'xyz   '
Resultado: 6
```

Listagem 7. Criação da SP para verificar a posição de uma string

```
CREATE PROCEDURE Pos (SubStr VARCHAR(100),
  Str VARCHAR(100))
RETURNS (Pos INTEGER) AS
/* 1 + SubStr-lenght + Str-length */
DECLARE VARIABLE SubStr2 VARCHAR(201);
DECLARE VARIABLE Tmp VARCHAR(100);
BEGIN
  IF (SubStr IS NULL OR Str IS NULL) THEN
  BEGIN
    Pos = NULL;
    EXIT;
  END
  SubStr2 = SubStr || '%';
  Tmp = '';
  Pos = 1;
  WHILE (Str NOT LIKE SubStr2 AND Str NOT LIKE Tmp) DO
  BEGIN
    SubStr2 = '_' || SubStr2;
    Tmp = Tmp || '_';
    Pos = Pos + 1;
  END
  IF (Str LIKE Tmp) THEN
    Pos = 0;
END
```

Listagem 8. Utilização da SP Pos

```
EXECUTE PROCEDURE Pos 'ab', 'abcdefghij'
Resultado: 1

EXECUTE PROCEDURE Pos 'cd', 'abcdefghij'
Resultado: 3

EXECUTE PROCEDURE Pos 'x', 'abcdefghij'
Resultado: 0
```

Podemos omitir "len = null;" porque as variáveis são inicializadas automaticamente como nulas. O comprimento será calculado contando inclusive os espaços à direita, conforme podemos visualizar nos exemplos da **Listagem 6**.

Função de posição (POS) em uma string em SP

Essa função retorna o índice do primeiro caractere de uma *substring* especificada (parâmetro *SubStr*) que ocorre em uma *string* dada (*Str*). Na **Listagem 7** temos a implementação da *Stored Procedure*.

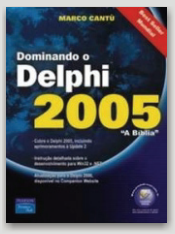
A variável *Tmp* é utilizada para interromper o laço quando o número de iterações for igual ao comprimento de *Str*. Como *SubStr* é usado no lado direito do operador LIKE, não poderá conter caracteres de substituição SQL, isto é "_" e "%" (a menos que utilizamos a cláusula ESCAPE).

Caso a *substring* não for encontrada, o valor de retorno é zero. Na **Listagem 8** temos exemplos de utilização da *Stored Procedure*.

Conclusão

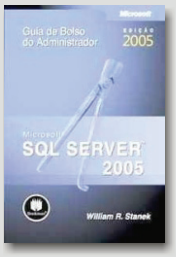
Vimos neste artigo algumas interessantes funções implementadas em forma de Stored Procedures, que podem ajudá-lo quando necessitar trabalhar com strings em aplicações com o banco de dados Firebird. ■

Destaque



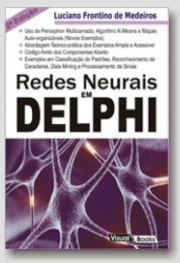
Dominando O Delphi 2005 - A Bíblia
Pearson / Prentice Hall
De: R$ 169,00
Por: R$ 160,
Até 6X de R$ 26,76 no Hipercard
Até 4X de R$ 40,14 no Visa e MasterCard

Sql Guia Prático 2ª Edição
Brasport
R$ 53,00



Sql Server 2005 - Guia de Bolso do Administra
Bookman
R$ 82,00

Firebird 2.0 - O Banco de Dados do Novo Milênio
Ciência Moderna
De: R$ 62,00
Por: R$ 58,90



Redes Neurais em D
2ª Edição
Visual Books
De: R$ 42,00
Por: R$ 39,90

Frete Grátis
Para todo o Brasil!
4x no Visa e Maste
e 6x no Hiperca
Outras facilidades de pagamento:
Tel: (11) 4062-5152
livrosdeprogramac

Novidades do Firebird 2.0

JORGE LUIS BUBLITZ
(bublitz@hotmail.com),
é desenvolvedor autodidata desde os 14 anos e formado em Administração de Empresas. Começou com BASIC nos micros de 8 bits, como o Apple II (TK-2000), TRS-80 (CP-500) e MSX (Expert). Desenvolveu em Clipper e em Turbo Pascal desde a versão 3. Desenvolve em Delphi desde a versão 1. Atualmente é funcionário concursado do Tribunal Regional Eleitoral de Mato Grosso e chefe da Seção de Análise e Desenvolvimento da Secretaria de Tecnologia da Informação, onde coordena e desenvolve diversos projetos Cliente/Servidor, Multicamadas, .NET e VoIP, todos em Delphi.

No dia 12 de novembro de 2006, o Projeto Firebird (*www.firebirdsql.org*) oficialmente liberou a tão esperada versão 2.0 do Firebird, durante a sessão de abertura da 4ª Conferência Internacional de Firebird que aconteceu em Praga, República Tcheca. Neste artigo veremos algumas das novidades dessa versão, cuidados com a migração e o que virá na próxima versão.

Uma nova versão

O Firebird 2.0 é o resultado de mais de dois anos de esforços de uma dedicada e variada comunidade de desenvolvedores e apoiadores ao redor do mundo. A nova versão traz uma série de melhorias que aumentam de forma significativa o desempenho, segurança e suporte a diversos idiomas e características de línguas (inclusive o *Collate* PT_BR) e outras novas funcionalidades da linguagem SQL.

O seu núcleo oferece uma plataforma de código muito mais robusta onde a re-arquitetura planejada para a versão 3.0 do Firebird (que será fundido com o banco *Vulcan*) terá bases sólidas. Algumas funcionalidades dessa versão são:

- Um total remanejamento das estruturas de índices para alavancar a desempenho e minimizar gargalos nas pesquisas;
- Várias limitações herdadas do código legado se foram, tais como o limite de 252 *bytes* por índice e os 30GB de limite de tamanho por tabela;
- Nova interface para conjuntos de caracteres internacionais através de *plugins* e melhorias no suporte a *Unicode*, entre uma série de correções e novas famílias de *collations*;
- Suporte às plataformas de 64 bits para AMD64 e Intel EM64T para Linux. Versões de 64 bits para Windows estão prontas para testes e serão lançadas em sub-releases;
- Intenso remanejamento da segurança do servidor, incluindo criptografia mais robusta das senhas (algoritmo *hash*) e tratamento interno de identificação de ataques de força bruta.

O algoritmo de encriptação/desencriptação de senhas era baseado na velha tradição Unix usando apenas os oito primeiros caracteres. Com o atual poder de computação, essas senhas eram facilmente quebradas utilizando o ataque de força bruta (tentando todas as combinações até oito caracteres);

• Suporte para tabelas derivadas conforme padrão SQL200x, incluindo alinhamento multi-nível e ligações com sub-queries;

• *Execute Block* para permitir que blocos de *SQL Procedural* (*PSQL*) sejam executados em sentenças SQL dinâmicas;

• Cursores explícitos em PSQL também disponíveis em sentenças *Execute Block*;

• *Sequence*, que foi introduzido como um sinônimo de *Generator*. *Sequence* é uma especificação SQL-99, enquanto que *Generator* é um conceito legado do InterBase. É aconselhável atualizar suas aplicações para utilizar *Sequence*;

• Tempo de espera opcional para conflitos de travamento, disponível tanto como argumento do comando *Set Transaction* quanto como parâmetro na API;

• Funcionalidades de *backups* incrementais de alto desempenho, com o novo utilitário *nbackup*;

• Completa re-arquitetura do protocolo de conexão local em Windows (*XNET*) para eliminar a instabilidade do protocolo legado (*IPServer*);

• Implementação completa da *Services API* em todas as plataformas.

Realmente o pessoal trabalhou muitas madrugadas nessa nova versão. O Firebird continua contando com duas versões, também chamadas de *arquiteturas*, que são: *Classic Server* e *Super Server*. Veja na **Tabela 1** as diferenças entre as mesmas.

Como podemos ver, nenhuma arquitetura é melhor que a outra em todos os aspectos. Agora vem a pergunta: "Qual a versão que devo usar?". Resposta:

• No Windows: *Super Server*, sempre;

• No Linux: Escolha uma ou outra, pois na prática você não verá grandes diferenças de desempenho.

Vale lembrar que você pode mudar de uma arquitetura para outra sem ter que mexer nas aplicações ou nas bases de dados. No Linux, os pacotes de instalação da arquitetura *Super Server* começam com *FirebirdSS* e da *Classic* por *FirebirdCS*. No Windows você escolhe a arquitetura durante o processo de instalação.

E ainda temos a versão *Embedded* (embutida), que é uma variação fantástica do servidor. Consiste em um servidor Firebird completo, composto por apenas alguns arquivos. É muito fácil de distribuí-lo, pois não há necessidade de instalação. Torna-se ideal para o uso em catálogos de CD-ROM, versões de avaliação de utilitários ou aplicações *standalone*.

ClubeDelphi PLUS

Acesse agora mesmo o Portal do Assinante ClubeDelphi e assista a uma vídeo-aula de Guinther Pauli que mostra como trabalhar com a versão Embedded do Firebird.

www.devmedia.com.br/articles/viewcomp.asp?comp=1734

Planejamento

Antes de desinstalar a versão anterior e instalar a nova, lembre-se de efetuar *backup* dos seus bancos. Além disso, você deverá também migrar a base de dados de usuários (*security.fdb*) do formato antigo para o novo (*security2.fdb*).

Isso é extremamente importante porque agora temos uma nova estrutura de arquivo ou *ODS* (*On-Disk Structure*), denominada *ODS11*. Dentre as novidades dessa nova versão, destacamos:

• O tamanho máximo de uma mensagem de exceção subiu de 78 para 1021 *bytes*;

• Descrição para *Generators*, *Tabelas*, *Campos*, *parâmetros* e outros (**Figura 1**);

• Introduzido o conceito de *ODS Type* para distinguir entre bancos InterBase e Firebird.

	Classic Server	Super Server
Processos	Cria um processo separado a cada cliente conectado, cada um com seu próprio cache. Usa poucos recursos se o número de conexões é baixo.	Um único processo serve a todas as conexões, usando-se threads. Espaço compartilhado de cache. Mais eficiente se o número de conexões simultâneas crescerem.
Conexões Locais	Permite um acesso direto e rápido arquivo de dados no Linux. O processo do cliente precisa ter direito de acesso ao arquivo.	No Linux, todas as conexões locais são tratadas como conexões de rede, usando localhost. Apenas o processo do servidor precisa ter direito de acesso ao arquivo de banco.
	No Windows, ambas as arquiteturas suportam conexões locais de maneira segura, sendo que apenas o processo do servidor precisa ter direito de acesso ao arquivo de banco.	
Multiprocessador	Suporte a SMP (Symmetric Multi-Processor ou Multi Processamento Simétrico). Melhor desempenho no caso de poucas conexões.	Sem suporte a SMP. No Windows o desempenho cai drasticamente, porque o SO transfere o processo de uma CPU para outra. Para prevenir isso, altere o parâmetro CpuAffinityMask no arquivo de configuração firebird.conf.
Guardian	Quando o Firebird é executado como uma aplicação, não é possível usar o Firebird Guardian. Isso só é possível no Windows 9x-ME.	Pode usar o Firebird Guardian tanto rodando como aplicação ou serviço.

Tabela 1. Diferenças entre as arquiteturas do Firebird

Se você abrir um banco criado com o Firebird 1.x, não esterarão disponíveis esses novos recursos. Então devemos fazer um *backup*, ainda com o Firebird 1.x, desinstalar a versão 1.x e só então instalar a versão 2.0.

Depois disso, você pode criar os novos bancos e restaurar os *backups*. O mesmo vale para o arquivo de usuários *security.fdb*, utilizando o script em *<installdir>/misc/upgrade/security*.

Pelos mesmos motivos devemos também atualizar a ferramenta de administração. A versão *2006.10.14 Personal* do *IBExpert* (*www.ibexpert.com*) da suporte ao Firebird 2.0.

A instalação do Firebird 2.0 não tem nenhuma novidade que necessite de comentários, sendo praticamente igual as versões anteriores. Vamos dar uma rápida olhada nas principais novidades no Firebird 2.0.

COMMENT

Agora podemos adicionar descrições ao banco, tabelas, campos, parâmetros e outros. Isso para documentação é excelente, pois quem nunca ficou em dúvida sobre o conteúdo de um campo e qual o objetivo dele?

Veja a sintaxe:

```
COMMENT ON DATABASE IS {'txt'|NULL};
COMMENT ON COLUMN tabela_view_nome.campo_nome IS
  {'txt'|NULL};
COMMENT ON <tipo> nome IS {'txt'|NULL};
```

Tipos onde podemos adicionar comentários: *Domain*, *Table*, *View*, *Procedure*, *Trigger*, *External Function*, *Filter*, *Exception*, *Generator*, *Sequence*, *Index*, *Role*, *Character Set*, *Collation* e *Security Class*.

Exemplo de comentário:

```
COMMENT ON COLUMN VENDAS.TIPO IS
  'Campo que armazena o tipo de pagamento';
```

SEQUENCE

Na prática *Sequence* funciona como *Generator* (no IBExpert ele fica armazenado no item *Generator*), só que seguindo o padrão SQL-99. Os comandos são:

```
CREATE SEQUENCE <nome>
DROP SEQUENCE <nome>
ALTER SEQUENCE <nome> RESTART WITH <valor_inicial>
NEXT VALUE FOR <nome>
```

Colattes para Língua Portuguesa (Brasil)

Agora temos dois *Collates case-insensitive/accent-insensitive* para a língua portuguesa do Brasil: WIN_PTBR (*Charset* WIN1252) e PT_BR (*Charset* ISO8859_1). Execute os comandos da **Listagem 1** e veja o resultado na **Figura 2**.

A boa novidade que esse *Collate* não diferencia maiúsculas de minúsculas e nem palavras acentuadas.

RETURNING em Insert

Em algumas situações, precisamos saber o valor que foi incluído em uma tabela, geralmente na chave primária gerada através de uma *Before Trigger*. Podemos utilizar então o seguinte código

```
INSERT INTO ... VALUES (...)
  [RETURNING <column_list> [INTO <variable_list>]]
```

Caso você queira que retorne os valores em variáveis, só podemos utilizar esse recurso em uma *Stored Procedure*. Vale lembrar que os valores atribuídos a campos em uma *After Tri-*

ClubeDelphi PLUS

Acesse agora mesmo o Portal do Assinante ClubeDelphi e assista a uma vídeo-aula de Luciano Pimenta que mostra como trabalhar com o IBExpert para banco de dados Firebird.

www.devmedia.com.br/articles/viewcomp.asp?comp=3082

Listagem 1. Exemplo de Comment, Sequence e Collate WIN_PTBR

```
CREATE TABLE CLIENTES (
  ID INTEGER NOT NULL,
  NOME VARCHAR(50) CHARACTER SET WIN1252
  COLLATE WIN_PTBR);

CREATE SEQUENCE ID_CLIENTE;

COMMENT ON SEQUENCE ID_CLIENTE IS
  'Sequence para controlar o Código do Cliente.';

INSERT INTO CLIENTES VALUES (
  NEXT VALUE FOR ID_CLIENTE, 'João');
INSERT INTO CLIENTES VALUES (
  NEXT VALUE FOR ID_CLIENTE, 'JOSÉ');
INSERT INTO CLIENTES VALUES (
  NEXT VALUE FOR ID_CLIENTE, 'joão');
INSERT INTO CLIENTES VALUES (
  NEXT VALUE FOR ID_CLIENTE, 'JOÃO');
INSERT INTO CLIENTES VALUES (
  NEXT VALUE FOR ID_CLIENTE, 'joao');

SELECT * FROM CLIENTES
WHERE NOME like 'joao%'
ORDER BY NOME;
```

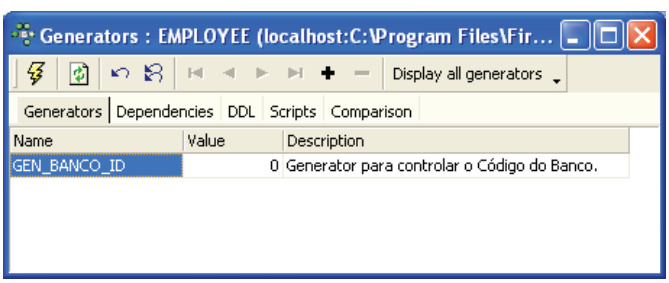


Figura 1. Descrição de um Generator

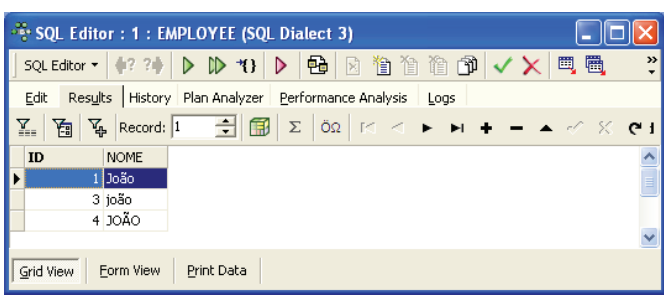


Figura 2. Exemplo de utilização de Sequence e o Collate WIN_PTBR

gger não serão atribuídos as variáveis. Veja o resultado do seguinte comando:

```
INSERT INTO CLIENTES VALUES (
  NEXT VALUE FOR ID_CLIENTE, 'Antônio')
RETURNING ID;
```

Veja na **Figura 3** o exemplo em execução no IBExpert.

Você deve estar se perguntando: "Como consegui sobreviver sem esse recurso até hoje?". Continuando com o exemplo anterior, vamos criar uma *Trigger* e uma SP para testar o recurso. Execute os comandos da **Listagem 2**.

Listagem 2. Testando a cláusula RETURNING

```
CREATE TRIGGER CLIENTES_BI FOR CLIENTES
ACTIVE BEFORE INSERT POSITION 0 AS
BEGIN
  IF (NEW.ID IS NULL) THEN
    NEW.ID = GEN_ID(ID_CLIENTE, 1);
END

CREATE PROCEDURE NOVO_CLIENTE (
  NOME_CLIENTE VARCHAR(50))
RETURNS (IDCLIENTE INTEGER) AS
BEGIN
  INSERT INTO CLIENTES (NOME)
  VALUES (:NOME_CLIENTE)
  RETURNING ID INTO :IDCLIENTE;
END
```

Quando executamos *NOVO_CLIENTE*, passando como parâmetro o nome a ser incluído, ela retorna o código que foi atribuído a esse novo cliente (**Figura 4**).

Vamos fazer um exemplo para mostrar como funciona no Delphi. Crie uma nova aplicação e adicione alguns componentes, conforme a **Figura 5**.

Temos um *DBGrid*, um *Edit* e um *Button*. Para acessar o banco, temos: um *SQLConnection*, o trio *SQLDataSet* + *DataSetProvider* + *ClientDataSet*, um *DataSource* e um *SQLStoredProc*. Configure o *SqlConnection* para acessar o banco que criamos anteriormente. No *SQLDataSet* aponte a propriedade *SQLConnection* para o *SQLConnection1* e o *CommandText* para:

```
select ID, NOME from CLIENTES order by NOME
```

No *DataSetProvider* aponte a propriedade *DataSet* para o *sqlCliente*. No *ClientDataSet* altere o *ProviderName* para *dspProvider* e *Active* para *True*. No *DataSource* aponte a propriedade *DataSet* para o *cdsCliente*. No *DBGrid* aponte a propriedade *DataSource* para o *dsCliente*.

No *SQLStoredProc* aponte a propriedade *SQLConnection* para *SQLConnection1* e *StoredProcName* para NOVO_CLIENTE. No evento *OnClick* do botão, digite o código da **Listagem 3**.

Listagem 3. Código do evento OnClick do botão de incluir

```
procedure TForm1.btnIncluirClick(Sender: TObject);
var
  cod_cliente: integer;
begin
  SQLStoredProc1.Params[0].AsString := Edit1.Text;
  SQLStoredProc1.ExecProc;
  cod_cliente := SQLStoredProc1.Params[1].AsInteger;
  ShowMessage('O código do novo cliente é: ' +
    IntToStr(cod_cliente));
  ClientDataSet1.Refresh;
end;
```

Execute o projeto, digite um nome qualquer no *Edit* e clique no botão *Incluir*. Veja o resultado na **Figura 6**.

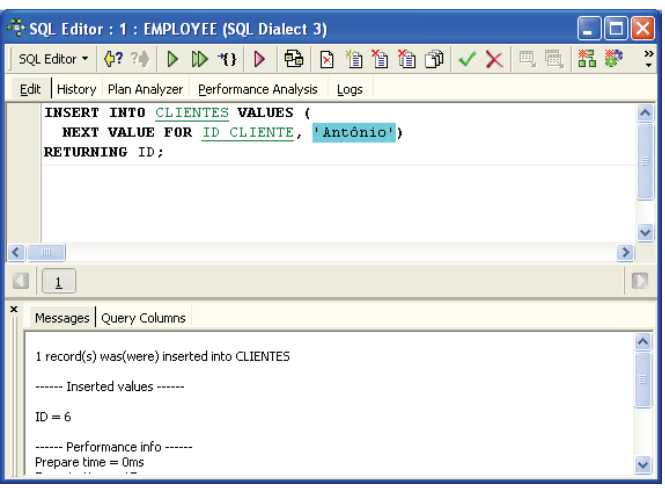


Figura 3. Retornando o valor da chave do registro recém adicionado

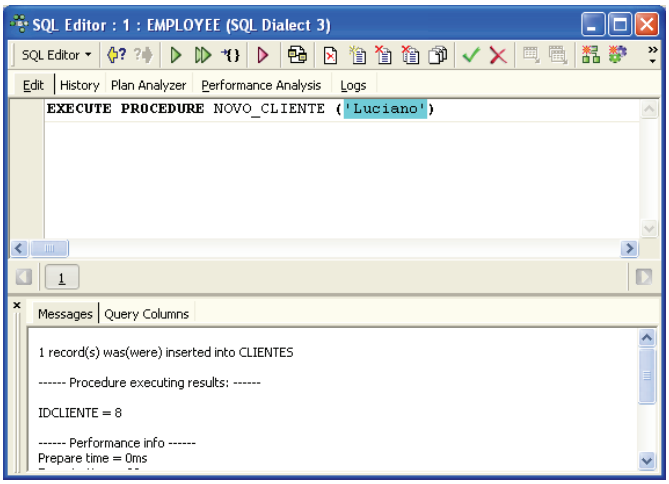


Figura 4. Retornando a chave adicionada, através de uma Stored Procedure

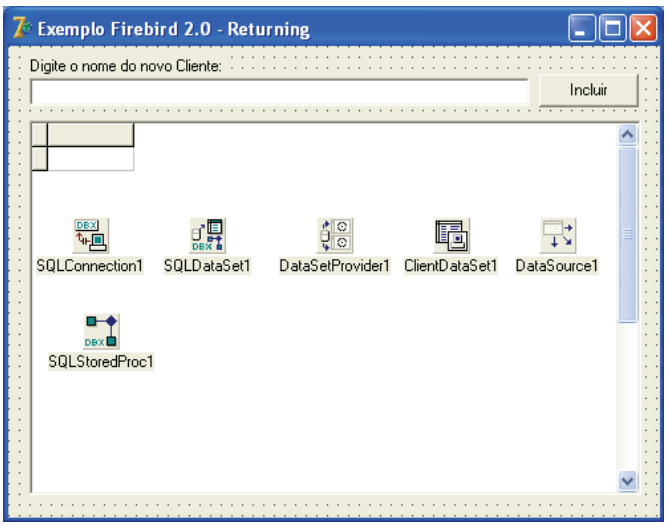


Figura 5. Aplicação para testar o Returning no Delphi

Nota: O *driver* testado nesse artigo é o do InterBase. Mas por questões de compatibilidade, sugiro a utilização de outro *driver*, mais específico para o Firebird, sendo que já existe *drivers* gratuitos que suportam o Firebird 2.0.

ROWS

Agora podemos determinar quais linhas de um *Select* queremos que sejam retornadas. Isso só era possível através de *Stored Procedures* ou através da aplicação. ROWS também está de acordo com padrão SQL. Pode ser utilizado tanto com *Select*, *Union*, Subquery, *Update* ou *Delete*:

```
SELECT | UPDATE | DELETE ...
[ORDER BY <expr_list>]
ROWS <expr1> [TO <expr2>]
```

Vejamos alguns exemplos: O código a seguir, seleciona todos os campos da tabela T1 juntamente com a tabela T2, ordenado pelo primeiro campo, mostrando as linhas (registros) de 10 até 20:

```
SELECT * FROM T1
UNION ALL
SELECT * FROM T2
ORDER BY 1
ROWS 10 TO 20
```

Outro exemplo no código a seguir, apaga os cinco primeiros registros da tabela T1, de acordo com a ordem da COL5:

```
DELETE FROM T1
ORDER BY COL5
ROWS 1 TO 5
```

Essa funcionalidade é ótima para paginarmos registros em uma aplicação Web.

Função IIF

Para os Clippeiros de plantão, essa função traz velhas recordações. Sua sintaxe é bem simples:

```
IIF(<condição>, <valor1>, <valor2>).
```

Ela é a forma abreviada do *Case*:

```
CASE
WHEN <condição> THEN <valor1>
ELSE <valor2>
END
```

Retorna o primeiro valor se a condição for verdadeira, senão retorna o segundo valor. Exemplo:

```
SELECT IIF(VAL > 0, VAL, -VAL)
FROM OPERATION
```

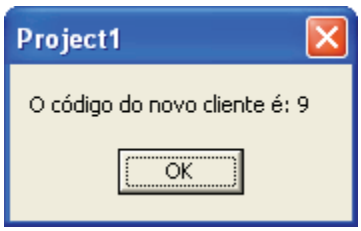


Figura 6. Retornando o código do registro incluido

E o futuro?

Para a nova versão 3.0 teremos os seguintes recursos:

- Ferramentas de monitoramento;
- Usuários no próprio banco (*Embedded users*);
- Tabelas temporárias;
- Funções em SQL;
- Queries recursivas;
- *Outer Joins* mais rápidos;
- Suporte a SMP na arquitetura *Super Server*;
- Suporte a funções e *procedures* externas.

Conclusão

A versão 2.0 do Firebird é o que podemos chamar de fase madura do produto. Melhorias em performance, novas funcionalidades e novos recursos, com certeza já entrou para a história da informática e o seu futuro é muito promissor. Suas principais características: banco pequeno (o arquivo de instalação para Windows tem 4,12 MB e ocupa 14 MB de disco), leve, que dispensa DBA's e, principalmente, seguro, estável e rápido.

Algumas pessoas pensam que o Firebird é um SGBD para ser usado apenas com pequenos bancos de dados ou poucas conexões. O FB é usado com grandes bancos de dados e considerável número de conexões. Um bom exemplo foi mostrado na *Softool'06* (*www.youtube.com/watch?v=XlF5ARsEZNw*), onde o *Avarda* (ERP russo) estava rodando com um servidor Firebird 2.0 Classic e um número médio de 100 conexões simultâneas, acessando um banco de dados de 120GB com 700 milhões de registros! O servidor era uma máquina bi-processada (2 CPUs - Dell PowerEdge 2950) com 6GB de RAM. Abraços e até a próxima. ■

ClubeDelphi PLUS

Acesse agora mesmo o Portal do Assinante ClubeDelphi e assista a uma vídeo-aula de Luciano Pimenta que mostra como trabalhar com o driver UIB dbExpress, especifico para o Firebird no curso on-line de uma aplicação completa client/server no Delphi 7.

www.devmedia.com.br/articles/viewcomp.asp?comp=1941

Links

Site Oficial

www.firebirdsql.org

IBExpert

www.ibexpert.com

FlameRobin (ferramenta de administração open source)

www.flamerobin.org

Complete sua coleção!

Feita para Desenvol

SQL magazine

Escolha 03 edições e pague **R$ 6,90** em cada revista.

apeamento Objeto acional. Aprenda a construir mapeador O/R em .NET.

arehouse Builder. Construa a data warehouse com a amenta ETL da Oracle-tel

ML. Aprenda passo a passo no construir um diagrama de uências.

imização. Influenciando o nizador de consultas.

21

- Normalização. Aprenda a organizar melhor o esquema de seu banco de dados

- PostreSQL. Saiba o que são rules e como utilizá-las.

- SQL Server 2000. Entenda os modelos de recovery.

- InterBase 7.5. Quando e porque migrar.

- Particionamento do Oracle. Perigos e armadilhas Parte II.

22

- Backup. Saiba como proteger seus dados no Oracle e no SQL Server.

- UML. Conheça algumas opções de ferramentas livres - Parte II.

- Homologação de dados. Saiba como avaliar a qualidade de seu modelo de dados.

- Especial. Acessando servidores Oracle a partir do SQL Server 2000.

23

- Proteja-se. Saiba como funciona o SQL Injection e aprenda a lidar com ele no SQL Server.

- Administrando o Oracle. Conheça o Oracle Enterprise Manager.

- Backup. Saiba como efetuar o restore de Bds no SQL Server.

- Modelagem. Utilizando padrões na elaboração do projeto de banco de dados.

24

- Java + BD + Web. Trabalh forma integrada com: MySQ Tomcat e Java com NetBean

- Oracle 10g. Aumente a disponibilidade de seu SGB com o automatic Storage Management.

- PostgreSQL. Veja na prátic como efetuar replicação ent servidores PostgreSQL.

ustering no SQL Server. Saiba a que serve e como funciona a tecnologia de redundância.

Expert. Configure e administre e de dados no Firebird e rbase.

aché. trabalhe com objetos etamente no banco de dados.

racle Discoverer. A prenda so a passo a trabalhar com a amenta de análise de dados da cle.

26

- Mineração de dados. Entenda como ocorre a descoberta de informações em base de dados.

- IBExpert. Administrando o InterBase/Firebird. Conheça algumas características avançadas.

- UML. Ferramenta de modelagem free com engenharia reversa.

- MySQL. Transforme-se em um especialista na otimização de consultas - Parte II.

27

- MySQL. Veja como rodar um banco de dados MySQL em várias máquinas ao mesmo tempo!

- ER/Studio. Conheça esta excelente ferramenta para modelagem de dados.

- SQL. Descobrindo associações entre pares de linhas.

- SQL Server. Veja como enviar e-mails anexos no SQL Server sem o uso do SQL Mail.

28

- Dúvidas frequentes em bancos de dados.

- Desafio SQL. Construindo um modelo para um jogo de futebol on-line.

- Index Tuning Wizard. Como produzir tunings automáticos.

- Erwin conheça esta ferramenta de modelagem E/R.

29

- Oracle FLASHBACK. Recupere informações para u determinado ponto do tempo.

- SQL Server. Descubra se vo esta trabalhando corretament com o SQL Server.

- InterBase / Firebird. Trabalhando com stored procedures e triggers.

www.devmedia.com.br/msdn/anteriores

Seu potencial. Nossa inspiração.™

Microsoft®

Novo Visual Studio® 2005. A diferença é óbvia.

Descobriu a diferença entre as duas imagens acima? Assim que começar a desenvolver, você descobrirá. O novo Visual Studio® 2005 possui mais de 400 novos recursos, como controles Web e Windows®, que reduzem tarefas entediantes e repetitivas, para que você possa focar a criação de códigos de qualidade. Saiba mais sobre o novo Visual Studio® 2005 acessando **www.msdn.com.br/visualstudio2005**

Microsoft® Visual Studio® 2005

© 2005 Microsoft Corporation. Todos os direitos reservados. Microsoft, Visual Studio, o logo do Visual Studio, Windows e "Seu potencial. Nossa inspiração" são marcas registradas da Microsoft Corporation nos Estados Unidos e/ou em outros países.